Hidden Markov Models And Their Applications In Stock Market Prediction

Jay Sharma
MMath Project
supervised by Dr Omar Lakkis

June 6, 2020

Abstract

Hidden Markov Models (HMM) are popular statistical tools for modelling different varieties of time-series based data. The models are very rich in mathematical structure and therefore have found useage across a wide array of disciplines. Their ability to convert the observable outputs that are emitted by real-world processes into predictable and efficient models makes them a viable candidate to be used for stock market analysis. The stock market has several interesting properties that make modeling non-trivial, namely volatility, time dependence and other similar complex dependencies. HMMs are suited to dealing with these complications as the only information they require to generate a model is a set of observations. In this project the first section attempts to methodically review the theory behind Hidden Markov Models in an easily understandable way, going through simple examples and using diagrams to make the three key algorithms more comprehensible. Once the theory has been clarified, the final third of the project presents an experiment that utilizes a Maximum a Posteriori HMM approach for forecasting closing stock prices for the next day given some historical data. This approach is tested on several stocks, and the accuracy of the predictions is determined by comparing the predicted values with the actual stock values by using the measure Mean Absolute Percentage Error (MAPE). The MAPE values found for all four stocks were below 3%, suggesting that HMMs are a valid tool for stock market prediction.

Contents

1 Introduction

Real-world processes typically produce a set of observable outputs which can be analysed in different ways. Despite the fact that the emissions for real-world processes are easily observed (audio in a recording, text in a book, etc), the underlying mechanism that models how a given process generates these observations is often unknown. Hidden Markov Models (HMMs) are stochastic processes that extend the theory of standard Markov models and provide us with a means to approximate the aforementioned underlying mechanisms behind real-world processes. The model that is approximated has many different potential uses, with prediction of future observations and the generation of new sequences of observations being amongst some of the most common. The theory behind HMMs was first introduced in the 1960's in a paper by Baum and his colleagues [3] and due to their broadly applicable nature they were quickly adapted for use in speech analysis, DNA-sequencing (bioinformatics), machine learning, vision recognition and more.

The stock market provides a rich source of observations and historical data that can be modelled as time-series'. Predicting stock values, even a fraction of a second in advance, can present enormous arbitrage profit opportunities, and so there is motivation in both industry and academia to find a way to predict stock prices. Stock price prediction is a difficult problem to solve, as there are many factors that can influence the movement of any given stock, including volatility, seasonality, and company decisions (e.g. mergers). Due to the fact that HMM's have been proven to be successful at predicting time dependent phenomena, it is a natural leap to assume that they may have some potential for predicting the stock market. Multiple academic papers have attempted to model stock market changes using HMMs, with the most notable findings being from research done by Gupta and Dhingra [11], Hassan [12], and Shi and Weigend [22]. This project follows on from these papers by performing an analysis of four stocks using code written in Python.

The first two thirds of the project outline all of the required theory behind Hidden Markov Models, whilst the final third focuses on a real-world application of the theory by running an experiment that attempts to predict the stock market using a program written in Python. Section 2 starts by explaining all of the probabilistic assumptions that are taken into account when outlining the algorithms and theory behind HMMs. It then goes on to provide a brief introduction to standard Markov processes and the Markov assumption, which is crucial to understanding how HMMs work. Section 3 explores the fundamental theory of Hidden Markov Models; how they are laid out in mathematical notation, the key differences between standard and hidden Markov models, the three problems that HMMs are used to solve and the solutions for these three problems. These solutions involve detailed explanations of the forward-backward algorithm, the Viterbi algorithm and the Baum-Welch algorithm. Finally, this theory is extended beyond discrete HMMs to consider the cases for which the observations are continuous, since an understanding of continuous observation-based HMMs is required for the experiment.

Section 4 explains how the stock market can be analyzed using Hidden Markov Models and outlines the methodology behind the experiment. Four different stocks were selected for analysis - Apple, Alphabet, Tesla and Amazon - and their data for a given period was split into training and testing data. A separate HMM was trained for each stock, using the relevant training data, and then the model was tested by seeing how closely it's predictions match the actual values in the test data. The main finding was similar to that found in previous research papers [11] [12] [22], namely that HMMs can predict stock market prices with high levels of accuracy. Finally, in section 5 the results are discussed and the paper is concluded.

2 Prerequisite Probability Theory

The mathematics behind Hidden Markov Models requires an understanding of some fundamental probability theory. The theorems outlined in this section form the basis that allows us to make certain claims about Hidden Markov Models and know that they are mathematically justified. All of the information below can be obtained from Sheldon Ross' book, "A First Course in Probability" [18].

2.1 Basic Probability theory

Given a finite sample space Ω, for each event E in Ω, we assume that a number $P(E)$ is defined. $P(E)$ represents the probability of the event E occurring. This probability satisfies the following three axioms:

Axiom 1. $0 \leq P(E) \leq 1$

Axiom 2. $P(\Omega) = 1$

Axiom 3. *For any sequence of mutually exclusive events E_1, E_2, ... (that is, events for which $E_i E_J = \varnothing$ when $i \neq j$),*

$$P(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} P(E_i)$$

The first and second axiom are self explanatory. The third axiom states that, for any sequence of mutually exclusive events, the probability of at least one of these events occurring is just the sum of their respective probabilities [18]. The third axiom can be written more simply as

$$P(X + Y) = P(X) + P(Y)$$

assuming that X and Y are mutually exclusive events in Ω.

Hidden Markov Models typically involve two or more probabilities at once, as well as conditional probabilities so we require several more theorems.

Definition 2.1. Joint probability If X and Y are two independent random variables, then the joint probability function is defined as:

$$P(X \cap Y) = P(X, Y) = P(X)P(Y)$$

Theorem 2.2. ***Conditional probability*** *For two random variables X and Y, the probability of X conditioned on Y is defined as:*

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

If X and Y are mutually exclusive, then we have that:

$$P(X|Y) = 0$$

If X and Y are independent, then:

$$P(X|Y) = P(X)$$

Corollary 2.3. ***Product rule*** *Using the definition of conditional probability, the product rule is:*

$$P(X, Y) = P(X|Y)P(Y) = P(Y|X)P(X)$$

Corollary 2.4. ***Chain rule*** *The chain rule is an extension of the product rule for the general case. Consider a collection of random variables from $X_1, ..., X_n$. To find the value of a member of the joint distribution, we can apply the definition of the conditional probability to obtain:*

$$P(X_1, X_2, ..., X_n) = P(X_1|X_2, ..., X_n)P(X_2|X_3, ..., X_n)...P(X_{n-1}|X_N)P(X_n)$$

Theorem 2.5. ***Bayes' Theorem*** *Bayes' theorem is an alternative method of calculating the conditional probability if the joint probability isn't known. Bayes' theorem is given by the following equation:*

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

given that X and Y are events and that $P(Y) \neq 0$

2.2 Standard Markov Models and the Markov property

Markov models (sometimes referred to as Markov processes) were discovered by Andrei Markov early in the twentieth century, with his first paper on the topic being published in 1906 [15]. Markov analysed $20,000$ Russian letters from Pushkins novel "Eugene Onegin" and created two states to observe, vowels and consonants. He studied the chain properties between these vowels and consonants and proved that the letters in text are not independent, and that the relationship between letters can be modeled by probabilities between 0 and 1. An example of the kind of results that he found is that the probability of two vowels in succession is 0.19. The relationship between vowels and consonants formed the basis for the first and most widely used Markov model, the Markov chain [15]. Currently, one of the most common uses of Markov chains is to model the likelihood of transitions between incredibly large sets of states, with well known examples of this being how Google models their search results or Spotify's music recommendation software. There is an abundance of literature on standard Markov models but this section will primarily focus on the aspects that are relevant to HMMs.

The simplest definition of standard Markov processes is that they are stochastic processes that satisfy the Markov property, which is included in the Markov chain definition below, and is given by (2.1). Stochastic processes are generically defined by Cinlar as any process describing the evolution in time of a random phenomenon [6]. Stochastic processes can either be continuous or be based on discrete-time. We will explore discrete-time processes.

A discrete-time stochastic process $\{X_t : t \geq 1\}$ on a countable set S is a collection of S-valued random variables defined on a probability space $(\Omega, \mathcal{F}, P)$. The P is a probability measure on a family of events $\mathcal{F}$ (a σ-field) on the event-space mentioned in the previous chapter, Ω. The countable set S represents the state space of the process and the value $X_t \in S$ is the state of the process at time t[19]. The variable t can also represent a parameter other than time, like length. The finite-dimensional distributions for a discrete-time stochastic process are given by

$$P(X_1 = i_1, ..., X_t = i_t), \qquad i_1, ..., i_t \in S, t \geq 1.$$

These probabilities uniquely determine the probabilities of all events of the process [19]. The following definition of a Markov Chain is similar to one seen in Serfozo's "Basics of Applied Stochastic Processes" [19].

Definition 2.6. Markov Chain A stochastic process $X = \{X_t : t \geq 1\}$ on a countable set S is a Markov Chain if, for any $i, j \in S$ and $t \geq 0$:

$$P(X_{t+1} = j | X_1, ..., X_t) = P(X_{t+1} = j | X_t), \tag{2.1}$$

$$P(X_{t+1} = j | X_t) = p_{ij}, \tag{2.2}$$

Where p_{ij} represents the probability that the Markov chain moves from state i to state j. These transition probabilities satisfy $\sum_{j \in S} p_{ij} = 1$, $i \in S$, and the matrix $P = [p_{ij}]$ is the transition matrix of the chain.

The first condition, (2.1) represents the Markov property, the condition required to label a stochastic process as a Markov process. It states that, at any time t, the next state in the process X_{t+1} is conditionally independent of the past $X_0, ..., X_{t-1}$ given the present state X_t. In other words, this means that given the present state of the process, the past states have no influence on the future - the next state is dependent on the past and present only through the present state.

The second condition (2.2) states that the transition probabilities do not depend on the time parameter t and therefore the Markov chain is "time-homogeneous". This means that the probability of any state transition is independent of time, Markov chains can be labelled as having stationary transition probabilities.

Combining these two conditions we see that for any discrete-time Markov chain, the probability of changing from one state to the next is only affected by the current state, not the past states or how much time has passed/the current time. The final point in the definition states that the sum of all transitions between states must equal 1, and therefore the transition matrix given by P is a stochastic matrix - all values in each row sum to 1. We can illustrate the Markov property with a simple example.

Example 2.7. *Simple Urn Replacement* We have an urn that contains one black ball and two white balls. For three consecutive days a ball is going to be drawn on each day, without replacement. Let t represent the days and X_t represent the result for a given day. Suppose that we know that the draw when $t = 2$ is white, but we don't know anything about the outcome of the first days draw X_1. We know that the probability of the ball being either white or black on the final day is 0.5, because there are only two possible outcomes for the experiment, which are shown in the table below.

Table 1: Outcomes for experiment

Day	Outcome 1	Outcome 2
X_1	Black	White
X_2	White	White
X_3	White	Black

However, if we know about the first days result, then we know exactly what will happen on the third day - if the first day was white then $P(X_3 = \text{black}) = 1$, etc. Our knowledge of X_1 will determine the prediction we make for the third day, and this means that this simple stochastic process does not have the Markov property. If the experiment was changed to replace the balls every other day, then it would satisfy the Markov property as the probability of a future state would only depend directly on the prior day.

The types of Markov Chains we have explained so far fall under the class of first-order Markov models, the most commonly used form of Markov models, because the probability of the next state only relies on the current state. Markov models can be extended beyond first-order to n-th order Markov models that require information about the previous n states to determine the probability of future states. I have defined an n-th order Markov chain below.

Definition 2.8. An nth-order Markov chain A stochastic process $X = \{X_t : t \geq 1\}$ on a countable set S is an n-th order Markov Chain if, for $n \geq 0$ and $t \geq 0$:

$$P(X_{t+1} = j|X_1, ..., X_t) = P(X_{t+1} = j|X_{t-n}, ..., X_n) \tag{2.3}$$

Increasing the order of a Markov model allows one to build more memory into the states. More memory in the states can be desirable since the values of time-series in the real world often appear to change randomly but may be reliant on information from history. In these cases, higher order Markov models might be required since today's value can depend not only on yesterday's value but also on further values in the past. A forecast model that utilises higher order Markov chains can therefore give a more accurate prediction of future states. Without computer programs it is difficult to model higher order Markov chains as the amount of calculations required can grow very large very quickly. The experiment in Section 4 uses Python to analyse a tenth order Hidden Markov Model.

Hidden Markov models are reliant on two assumptions, the Markov assumption and the independence assumption. The Markov assumption is used to describe a model where the Markov property is assumed to hold [5] and the independence assumption is another way of saying that the models are assumed to be time-homogeneous. In other words, both (2.1) and (2.2) are assumed to be true for hidden Markov models. With an understanding of these two conditions, we are now ready to define HMMs.

3 Hidden Markov Models

The theory behind Hidden Markov Models was first introduced by Leonard E. Baum and his colleagues in the late 1960's [1]. After realising the potential applications of Hidden Markov Models in several different fields, Baum went on to write more papers throughout the 70's with hopes that HMM techniques could be applied to the stock market and weather prediction [4]. The flexibility provided by the "hidden" aspect of HMM's has allowed them to be applied to both of those fields, along with many more, including speech recognition, machine learning, handwriting-to-text translation and bioinformatics [17] [14]. Although they have now been around for over 40 years, the most comprehensive text on Hidden Markov Models is still Rabiner's seminal 1989 paper [17], which provides readers with an in depth understanding of how HMMs work, and their applicability to the speech recognition methods of the time. The bulk of the theory behind Hidden Markov Models explored in the first section of this chapter comes from this paper, alongside papers from Stamp, Blunsom and Poritz [20] [5] [16].

3.1 Definition and notation

A normal Markov chain has one state transition matrix, for which the probabilities are known, that is used to model a singular stochastic process. In contrast, a Hidden Markov model is defined as a doubly embedded stochastic process with an underlying Markov process modelling the states, that is not observable (hence "hidden"), but can only be observed through another set of stochastic processes that produce a sequence of observations. The state space of the hidden variables is typically discrete, but the observations can be either discrete (often generated by a categorical distribution) or continuous (often generated by a Gaussian distribution). In some cases, the sequence of observations (or emissions) is the only data available, and is used to infer what the likely "hidden" state sequence was, the state transition probabilities, and the likelihood of a given state producing a given observation. In other cases, there is already a sufficiently accurate HMM λ, so the model can be used as a generator to produce an observation sequence O from hidden states.

HMMs are capable of predicting and analyzing time-based phenomena. In terms of real world applications, Hidden Markov Models can be used to detect the underlying hidden states (and their transition probabilities) behind large sets of observed data, such as daily stock prices or historical weather data. Once the model has been trained so that the transition probabilities between the hidden states are fairly accurate, it can then be used to predict future observation sequences based on currently available information. HMM's can also be used to test whether our presumed model for the states is accurate, we might have assumed there are more (or less) hidden states than there actually are, or gotten the transition probabilities wrong. Whilst the verbal definition above provides some clarity on Hidden Markov Models, they are best understood through the use of examples and an exploration of the notation.

Hidden Markov Models are characterised by the following notation:

1. N represents the number of states in the model. N tells us the length of the state alphabet.

2. M represents the number of distinct observations that can be emitted by the hidden states. In other words, M is the size of the observation alphabet.

3. T represents the total length of our observation sequence, our final observation occurrs when $t = T$.

4. S is the state alphabet set, it lays out all the distinct states for the Markov process:
$$S = (s_1, s_2, ..., s_N) \tag{3.1}$$

5. V is the set of possible observations:
$$V = (v_1, v_2, v_3, ..., v_M) \tag{3.2}$$

6. We define Q to be our fixed (hidden) state sequence of length T:
$$Q = (q_1, q_2, ..., q_T) \tag{3.3}$$

7. The corresponding sequence of observations of length T is given by O:
$$O = (o_1, o_2, ..., o_T) \tag{3.4}$$

8. The matrix A is an $N \times N$ matrix that represents the transition array for the hidden states. Each entry a_{ij} stores the probability that the next state is s_j, given that the current state is s_i:
$$A = [a_{ij}], a_{ij} = P(q_{t+1} = s_j | q_t = s_i), \qquad 1 \leq i, j \leq N \tag{3.5}$$
The matrix A has the same properties as the matrix P from 2.6, it is row stochastic (i.e. $\sum_{s_j \in S} a_{ij} = 1$, $s_i \in S$) and the probabilities a_{ij} are independent of time t, where $0 \leq t \leq T$.

9. The matrix B is an $N \times M$ matrix that represents our observation array (emission matrix). The matrix stores the probability of an observation k, where $k \in V$, being produced by the state s_j at time t:
$$B = [b_j(k)], b_j(k) = P(o_t = k | q_t = s_j), \qquad 1 \leq j \leq N, 1 \leq k \leq M \tag{3.6}$$
The matrix B is similar to A in that it is row stochastic and that the probabilities $b_j(k)$ are independent of time t.

10. π is the initial state distribution:

$$\pi = [\pi_i], \pi_i = P(q_1 = s_i) \qquad 1 \leq i \leq N \tag{3.7}$$

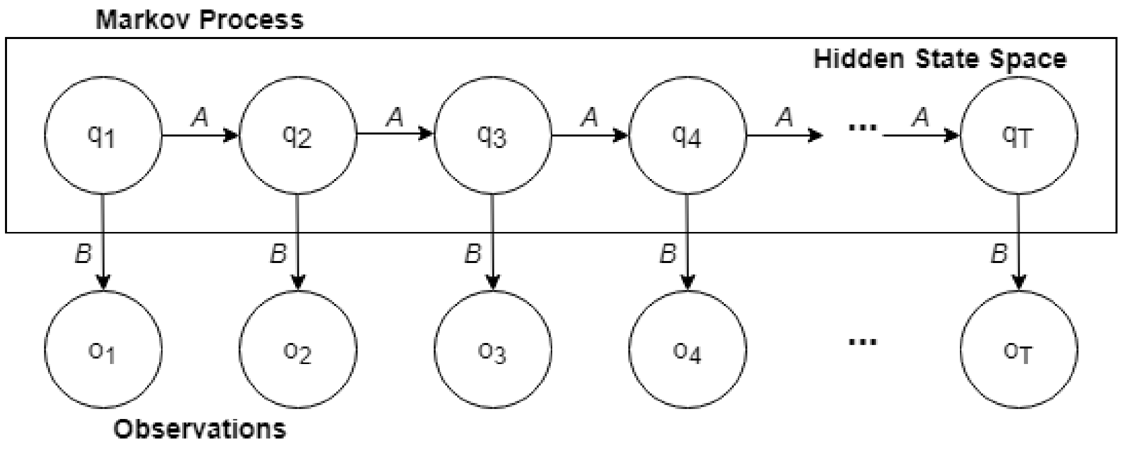
The initial state distribution is a $1 \times N$ matrix that describes the probability of the process beginning in each given state from our state alphabet S.

11. Finally, the formal definition of a Hidden Markov Model is given by λ, where:

$$\lambda = (A, B, \pi) \tag{3.8}$$

Once we have estimated our model λ (3.8) it can used as either a generator for a new sequence of observations, or as a model for how a given observation sequence was generated by an appropriate HMM. A graphic representation of a generic Hidden Markov Model can be seen in Figure 1. The figure follows the exact notation laid out above. The Markov process is contained within a rectangle to indicate that it is hidden from us. Within the hidden state space we have a sequence of states that changes over time, beginning at q_1 and ending at q_T. The hidden Markov process transitions from one state to the next via the A matrix, and is only reliant on the current state. The only data that we can observe is the sequence of observations $O = (o_1, o_2, o_3, o_4, ..., o_T)$, which are related to the hidden states of the Markov process by the emission matrix B.

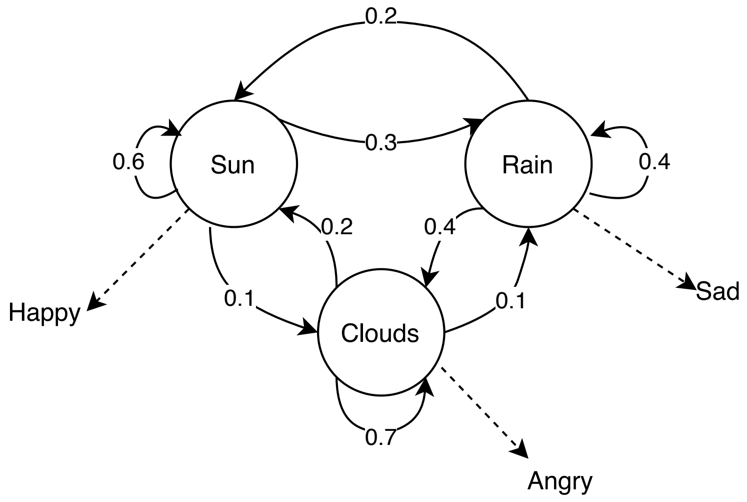
Figure 1: Generic Hidden Markov Model



3.2 A simple example

The best way to develop our understanding of HMM's and expand on the difference between them and standard Markov models is through an example. I've created two simple examples that allow us to deepen our understanding of the notation in a

Figure 2: Standard Markov Model



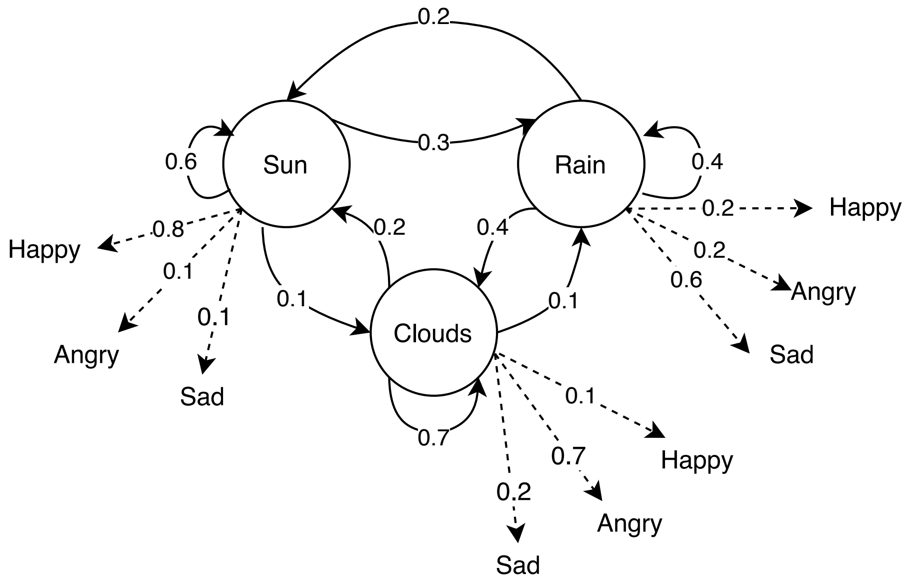
straightforward way. First we look at a standard weather based Markov chain - our states represent the weather and our observations are given by mood. Suppose that we have a friend John that lives in a different country whom we call everyday. Also suppose that we have some rough idea of the state transition probabilities by having looked at the historical weather data for his country. We know John well enough to know that his mood during this daily call is determined by the state of the weather - if it is sunny, he's happy, if it's rainy, he's sad, and if it's cloudy he's angry. Therefore we have $N = 3$ and $M = 3$, i.e. we have three states and three observations. Our state alphabet is given by $S = (R, S, C)$ where "Rain" = R, "Sun" = S and "Clouds" = C. The observation alphabet is given by $V = (Happy, Angry, Sad)$. The Markov chain that models this process is shown in Figure 2.

For this example the observations are tied directly to the underlying state, so even though we cannot see the weather in his country, it is still a normal Markov model since John's mood indirectly informs us of the current weather conditions. Given a sequence of observations, e.g $O = (Angry, Angry, Sad)$, we can easily verify that the state sequence that produced those observations was: $Q = (C, C, R)$, and that the probability of the sequence is simply the product of the transitions - $0.7 \times 0.7 \times 0.1 = 0.049$. Our state transition matrix A for this model is given by:

$$A = \begin{array}{c c} & \begin{array}{ccc} R & S & C \end{array} \\ \begin{array}{c} R \\ S \\ C \end{array} & \begin{pmatrix} 0.4 & 0.2 & 0.4 \\ 0.3 & 0.6 & 0.1 \\ 0.1 & 0.2 & 0.7 \end{pmatrix} \end{array} \tag{3.9}$$

The matrix A is row stochastic since the probabilities in each row sum to 1. The matrix shows the probability of transitioning from one state to the next, for example

Figure 3: Weather based Hidden Markov Model



$a_{11} = 0.4$ and represents the probability that it rains tomorrow, given that it rained today.

Figure 3 describes how the previous model can be extended into a Hidden Markov Model. The key difference between the two is that observation's of John's mood via calls no longer tell us the exact weather conditions for that day, and thus the state sequence is hidden from us. In the new model, John's mood is no longer directly tied to specific states but varies with some probability. In other words, all possible observations can be emitted by each state. This makes the model more expressive and more realistic - in reality it is unlikely that the weather would have a 1 to 1 cause and effect relationship with John's mood. This is also indicative of how, in a general sense, HMM's can be more practical when analysing real world data sets since the relationship between states and emissions can be complex. The observation array for figure 3 is given by B where:

$$B = \begin{array}{c|ccc} & \scriptstyle Happy & \scriptstyle Angry & \scriptstyle Sad \\ \scriptstyle R & 0.2 & 0.2 & 0.6 \\ \scriptstyle S & 0.8 & 0.1 & 0.1 \\ \scriptstyle C & 0.1 & 0.7 & 0.2 \end{array} \tag{3.10}$$

The matrix B is also row stochastic and displays the likelihood of each observation occurring from a given state. For example, the probability that John is happy on a sunny day is given by $b_S(Happy) = 0.8$, following the notation laid out in (3.6). To complete the HMM, let's assume that, again using historical weather data, we've calculated our initial state distribution π_w (w for "weather"), given by:

$$\pi_w = \begin{pmatrix} 0.1 & 0.5 & 0.4 \end{pmatrix} \tag{3.11}$$

The 1×3 matrix represents the probability of the state sequence starting in each of the 3 states, the first column being for rain, the second being sun and the third being clouds. With this initial state distribution, the hidden Markov model is complete. Since it is our first model, we can denote it as λ_1:

$$\lambda_1 = (A, B, \pi_w) \tag{3.12}$$

When attempting to model a stochastic process with HMM's we may have a set of different potential models (λ's) despite only having one fixed observation sequence. For example, in the case above, we may have a winter model and a summer model. The winter model could be based on the states snow and hail, with the summer model being made up of the states above - sun, rain and clouds. Beyond changing the actual presumed states, variation amongst models can also come from differing amounts of states, an individual could come up with several potential models with vastly different amounts of states. In a situation like this for which there are multiple different models we would want to determine which model is the most accurate for our given observation sequence, i.e. we want to find the greatest value of $P(O|\lambda)$. The most straightforward way of doing this is through enumerating every possible state sequence of length T [17].

The probability of the observations O given specific state sequence Q (defined in 3.3) and model λ follows Rabiners work [17] and is given by:

$$P(O|Q,\lambda) = \prod_{t=1}^{T} P(o_t|q_t,\lambda) = b_{q_1}(o_1) \times b_{q_2}(o_2) \; ... \; b_{q_T}(o_T) \tag{3.13}$$

We have assumed statistical independence for the observations. The probability of the state sequence Q occurring given our model λ is defined as:

$$P(Q|\lambda) = \pi_{q_1} \times a_{q_1 q_2} \times a_{q_1 q_2} \; ... \; a_{q_{T-1} q_T} \tag{3.14}$$

The joint probability of O and Q, which represents the probability that O and Q occur simultaneously, is simply the product of $P(O|Q,\lambda)$ and $P(Q|\lambda)$, i.e.:

$$P(O,Q|\lambda) = P(O|Q,\lambda)P(Q|\lambda) \tag{3.15}$$

The probability that we are looking for, $P(O|\lambda)$, is obtained by summing the joint probability in (3.15) over all possible state sequences Q, and is defined by:

$$\begin{aligned} P(O|\lambda) &= \sum_{Q} P(O|Q,\lambda)P(Q|\lambda) \\ &= \sum_{q_1 \cdots q_T} \pi_{q_1} b_{q_1}(o_1) \times a_{q_1 q_2} b_{q_2}(o_2) \; ... \; a_{q_{T-1} q_T} b_{q_T}(o_T) \end{aligned} \tag{3.16}$$

Equation (3.16) can be expounded upon with a verbal explanation. We start at time $t = 1$ in state q_1 with probability π_{q_1}. State q_1 emits the observation symbol o_1 with probability $b_{q_1}(o_1)$. One unit of time passes and we go from t to $t+1$ ($t = 2$). We transition from state q_1 to state q_2 with probability $a_{q_1 q_2}$. State q_2 generates the symbol o_2 with probability $b_{q_2}(o_2)$ and the process continues until time T. The number represented by $P(O|\lambda)$ determines how likely it is that our assumed model λ was responsible for a given set of observations, i.e. how accurate our hypothetical HMM is.

Using our current example model λ_1, we can calculate the likelihood that the model produced a specific sequence of observations. Using the same observation sequence from the standard Markov model, we have $O = (Angry, Angry, Sad)$. Using the method outlined in (3.16) we would have to calculate the probability of O occurring given each possible state sequence, and then sum them up. Define our first possible state sequence by $Q_1 = (S, S, S)$, and so the calculation to find the probability that the observations were the result of Q_1 and λ_1 is given by:

$$\begin{aligned} P(O|Q_1) &= \pi_2(b_S(Angry)) \times a_{22}(b_S(Angry)) \times a_{22}(b_S(Sad)) \\ &= 0.5(0.1) \times 0.5(0.1) \times 0.5(0.1) = 0.15 \end{aligned} \tag{3.17}$$

This process would need to be repeated for all possible (3^3) state sequences and then all the resulting probabilities would need to be summed together to find the probability that our model λ_1 generated O. This requires exponentially more and more calculations as T grows, with $2TN^T$ calculations being required for any given model with N states over time T [17]. This is both time consuming and computationally inefficient. A more efficient method for calculating the likelihood that a specific model produced a specific sequence of outputs exists and is known as the forward algorithm. The forward algorithm solves the first of the three core problems that HMM's seek to solve, so we will explore those before moving on to their solutions.

3.3 The three problems

There are three key problems that users of hidden Markov models may be required to solve.

1. The Evaluation Problem: Given a model λ and a sequence of observations $O = (o_1, o_2, ..., o_T)$ what is the probability that the model generates the observations $P(O|\lambda)$. The efficient solution to this problem is given by the forward algorithm which was first shown by Baum et al in 1970 [4].

2. The Decoding Problem: Given a model λ and a sequence of observations $O = (o_1, o_2, ..., o_T)$, what is the most likely state sequence $S = (s_1, s_2, ..., s_i)$ in the model that produces our sequence of observations. This can be tackled in 2 ways, finding the most likely state at each time or finding the highest

scoring (most likely) overall sequence using the Viterbi algorithm, which was found by Andrew Viterbi in 1967 [21].

3. The Learning Problem: Given a model λ and a sequence of observations $O = (o_1, o_2, ..., o_T)$, how can we adjust the parameters of the model λ to maximise the joint probability $\prod_O P(O|\lambda)$, i.e. to train the model to best characterise the states and observations [20].

In some cases we aren't required to solve all 3 of the problems. Consider some software that converts handwritten characters into readable typed text. Our first step in designing this software would be to use the solution to problem 3 to train a HMM, let's call it λ_A to recognise the letter "A". Once we believe our model has received sufficient training, we can begin to train another HMM, which we call λ_B, to recognise the letter "B". Then, if we're given a random hand written character, we can solve problem 1 for each model to find out whether this random character is more likely to be "A" or "B" or some other letter that the model hasn't learnt to recognise yet. This is an example of a case for which we wouldn't be required to solve problem 2 to optimise our HMM.

3.4 Forward-backward algorithm

The solution to the first problem is given by the forward algorithm, which was first shown by Baum et al in 1970 [4]. Let $\lambda = (A, B, \pi)$ be the full set of parameters. We have already considered the inefficient method of calculating $P(O|\lambda)$ so now we consider the efficient method, the forward algorithm. This algorithm is characterised by the forward variable $\alpha_t(i)$ which is given by:

$$\alpha_t(i) = P(o_1, o_2, ..., o_t, q_t = s_i|\lambda) \tag{3.18}$$

In words, $\alpha_t(i)$ represents the joint probability that the partial sequence of observations until time t is given by $(o_1, o_2, ..., o_t)$ and that the state of the HMM at time t is s_i, given the model λ. Our forward variable $\alpha_t(i)$ is solved for inductively in three steps.

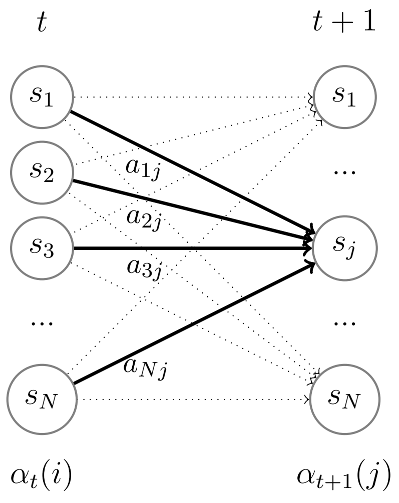
Definition 3.1. Forward algorithm

1. Initialisation:
$$\alpha_1(i) = \pi_i b_i(o_1) \quad 1 \leq i \leq N \tag{3.19}$$

2. Induction:
$$\alpha_{t+1}(j) = \Big[\sum_{i=1}^{N} \alpha_t(i)\alpha_{ij}\Big] b_j(o_{t+1}), \quad 1 \leq t \leq T-1 \text{ and } 1 \leq j \leq N \tag{3.20}$$

3. Termination:
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{3.21}$$

Figure 4: Trellis diagram illustrating the induction step of the forward algorithm. Displays the sequence of operations required for the calculation of the forward variable $\alpha_j(t+1)$



Step 1 initializes the forward probabilities as the joint probability of state s_i and initial observation o_1 at time $t = 1$.

Step 2, the induction step, is the core of the forward algorithm which drives the calculation forward and is best captured by a drawing, given by Figure 4. The left side of the figure displays all the possible states, s_i, $1 \leq i \leq N$, at time t. For each state i, the variable $\alpha_i(t)$ stores the probability of arriving in that state having observed the observation sequence, $O = (o_1, o_2, ..., o_t)$, up until time t. The figure shows how state s_j can be reached at time $t+1$ from the N possible states via the bold lines. Each bold line is labelled by the probability a_{ij} from our state transition matrix A to demonstrate the probability that state i transitions to state j from time t to time $t+1$. Summing the product $\alpha_t(i)a_{ij}$ across all N states results in the probability of s_j at time $t+1$, having taken into consideration all the accompanying previous partial observations. Once this is done and s_j is known, we see that $\alpha_{t+1}(j)$ is simply calculated by accounting for the observation o_{t+1} being emitted by state j, i.e by multiplying our summed probability by the product $b_j(o_{t+1})$. The computation given by (3.20) is then repeated for all states j, where $1 \leq j \leq N$, for a given t, and then iterated over for $t = 1, 2, ...T-1$.

Step 3 follows intuitively and gives the desired calculation of $P(O|\lambda)$ by summing up the final forward variables $\alpha_T(i)$. By definition the terminal forward variables are given by:

$$\alpha_T(i) = P(o_1, o_2, ..., o_T, q_T = s_i|\lambda) \tag{3.22}$$

This makes it clear that $P(O|\lambda)$ is the sum of these $\alpha_T(i)$'s. By caching these α

values, the forward algorithm greatly reduces the total amount of computations required. The forward algorithm only requires around N^2T multiplications, compared to the naïve approach given by 3.16 which requires $2TN^T$ multiplications [20].

The second part of the forward-backward algorithm is the analogous backward algorithm [4] [17], which is the reverse of the forward algorithm. The backward algorithm isn't specifically used to solve Problem 1, but it is used in the solutions for Problem 2 and Problem 3 and it naturally follows from the forward algorithm so we define it in this section. Define a backward variable $\beta_t(i)$ given by:

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, ..., o_T | q_t = s_i, \lambda) \tag{3.23}$$

This backwards variable represents the probability of the partial observation sequence $(o_{t+1}, o_{t+2}, ..., o_T)$ occurring, given the model λ and that the state at time t is s_i. The variable $\beta_t(i)$ is solved for inductively in a similar manner to the forward variable [17], except there are only two steps.

Definition 3.2. Backward algorithm

1. Initialization:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{3.24}$$

2. Induction:

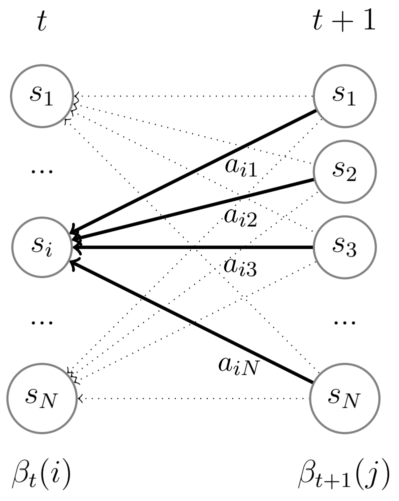
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \quad 1 \leq t \leq T-1, \ \ 1 \leq i \leq N \tag{3.25}$$

The $\beta_t(i)$ is computed recursively, with the first step arbitrarily defining $\beta_T(i)$ to be 1 for all i. The second step is displayed in Figure 5. The diagram shows that in order to have been in state s_i at time t, accounting for the observation sequence from time $t+1$, we have to consider all possible states s_j at time $t+1$, accounting for the transition probabilities from s_i to s_j which, are given by the arrows which are labeled a_{ij}. We also have to account for the observation o_{t+1} being emitted from state s_j (i.e. $b_j(o_{t+1})$) and then finally take into consideration the remaining partial observation sequence from state s_j, which is given by $\beta_{t+1}(j)$. The resultant $\beta_t(i)$ is the probability that the partial observation sequence given by $O = (o_{t+1}, o_{t+2}, ..., o_T)$ occurred, given the model and that the current state is s_i. The backwards variable is an essential part of solving problems 2 and 3.

3.5 Choosing the best state sequence

Unlike the first problem, the decoding problem (Problem 2) has multiple solutions. The aim of decoding a HMM is to discover the hidden state sequence S that was most likely to have produced a given observation sequence O. The need for multiple solutions stems from the fact that "most likely" can be interpreted in different ways.

Figure 5: Trellis diagram illustrating the induction step of the backward algorithm. Displays the sequence of operations required for the calculation of the backward variable $\beta_i(t)$



One possible interpretation is that the "most likely" - or optimal - state sequence is composed of the states q_t which are individually most likely at each time t where $1 \leq t \leq T$. This interpretation would rely on maximizing the expected number of correct individual states. Another possible interpretation of the "most likely" state sequence would be to find the highest scoring (most probable) overall path. This interpretation requires us to use a dynamic programming algorithm known as the Viterbi algorithm that is similiar to the forward algorithm, with the major difference being that it maximises the transition probabilities at each time t instead of summing them.

First we show how to find the optimal state sequence when considering which states are individually most likely at each time t, as shown in Stamp's work [20]. We define the variable $\gamma_t(i)$ as follows:

$$\gamma_t(i) = P(q_t = s_i | O, \lambda) \tag{3.26}$$

In words, (3.26) represents the probability that the current state (at time t) is s_i, given the model λ and the observation sequence O. We know that $\alpha_t(i)$ measures the relevant probability up to time t and accounts for the partial observation sequence $(o_1, ..., o_t)$, and $\beta_t(i)$ measures the relevant probability after time t, accounting for the rest of the observation sequence $(o_{t+1}, ..., o_T)$. Using this information, and Bayes theorem, we can express $\gamma_t(i)$ in terms of the forward and backward variables:

$$\gamma_t(i) = \frac{P(q_t = s_i, O|\lambda)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_(t)(i)}{\sum\limits_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{3.27}$$

Recall that the denominator $P(O|\lambda)$ was given in equation (3.21). The probability $P(O|\lambda)$ is used as a normalization factor to turn $\gamma_t(i)$ into a probability measure that satisfies:

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \tag{3.28}$$

The argmax function is commonly used in machine learning to find the argument that produces the maximum value from a specific function. Once we've found $\gamma_t(i)$, we use argmax to solve for the individually most likely state q_t at time t:

$$q_t = \underset{1 \leq i \leq N}{argmax}[\gamma_t(i)], \quad 1 \leq t \leq T \tag{3.29}$$

Equation (3.29) chooses the most likely state for each time t, and we hope to be able to sum across all t from $t = 1$ to $t = T$ to find the most likely state sequence that produced a set of observations $(o_1, ..., o_T)$ and thus solve problem 2. However, since we calculated the most likely individual state at each time as opposed to calculating the single most likely state sequence the utility of this method is entirely reliant on the HMM being fully connected. Until now, all the example HMM's that we have considered have been full connected - each state can reach any other state within one time step. This is not a pre-requisite for a stochastic process to be a hidden Markov model, and so there are many cases of HMM's for which there are state transitions with zero probability ($a_{ij} = 0$ for some i and j). This would mean that the equation (3.29) has the potential to find a state sequence that is invalid since it only calculates the most likely state at each time without taking into consideration the probability of sequences of states occurring. To prevent this issue from occurring, we use the Viterbi algorithm to find the single best state sequence. Finding the single best state sequence can also be viewed as trying to maximize $P(Q|O, \lambda)$ which is equivalent to maximizing $P(Q, O|\lambda)$ [17].

The Viterbi algorithm originated in Andrew Viterbi's 1967 paper [21] and was initially proposed as a decoding algorithm for convolutional codes over noisy digital communication links. The following formal definition of the Viterbi algorithm borrows from Viterbi's original paper and Forney's 1973 paper that applied the Viterbi algorithm to discrete-time finite-state Markov processes [21] [8].

Given some observed sequence $O = (o_1, o_2, ..., o_T)$ we seek to efficiently compute the state sequence $Q = (q_1, q_2, ..., q_T)$ that has the highest conditional probability given O and the model λ. That is, we want to find:

$$\underset{Q}{\text{argmax}} P(Q|O, \lambda) \tag{3.30}$$

We define the quantity $\delta_t(i)$ as:

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1 q_2 ... q_t = s_i, o_1 o_2 ... o_t|\lambda) \tag{3.31}$$

The variable $\delta_t(i)$ therefore represents the maximum probability (the highest scoring path), having taken into consideration all possible paths, of reaching state s_i at time t having observed the partial observation sequence $(o_1, o_2, ..., o_t$. In other words, the probability $\delta_t(i)$ is the highest scoring path at time t, which accounts for the first t observations and ends in state s_i. By induction, we see that the probability for the highest scoring path that ends in state s_j at time $t + 1$ is given by:

$$\delta_{t+1}(j) = [\max_{1 \leq i \leq N} \delta_t(i) a_{ij}] \times b_j(o_{t+1}) \tag{3.32}$$

To retrieve the state sequence, the Viterbi algorithm relies on storing the arguments that maximized equation (3.32) for each t and j. These arguments that result in the highest probability at each time are stored in the array $\psi_t(j)$. These stored arguments are called back pointers. These back pointers are stored during the recursion step and are used backtrack, after the algorithm has terminated, and find the optimal state sequence. The complete procedure for finding the most probable state sequence can now be seen in the formal definition of the Viterbi algorithm which is given below.

Definition 3.3. Viterbi algorithm

1. Initialization:

$$\delta_1(i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N \tag{3.33}$$

$$\psi_1(i) = 0 \tag{3.34}$$

2. Recursion:

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(o_t), \quad 2 \leq t \leq T, \;\; 1 \leq j \leq N \tag{3.35}$$

$$\psi_t(j) = \operatorname*{argmax}_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}], \quad 2 \leq t \leq T, \;\; 1 \leq j \leq N \tag{3.36}$$

3. Termination:

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)] \tag{3.37}$$

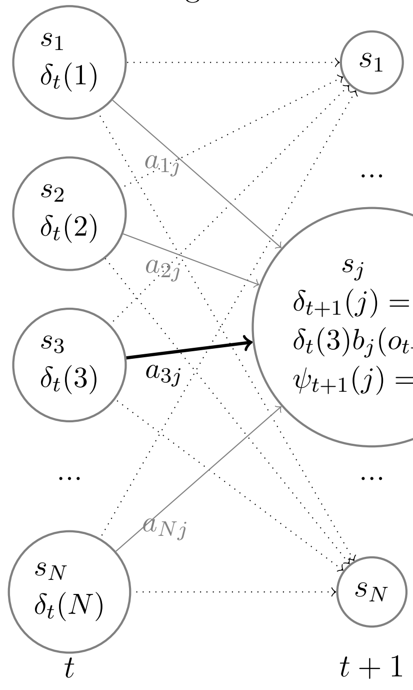
$$q_T^* = \operatorname*{argmax}_{1 \leq i \leq N} [\delta_T(i)] \tag{3.38}$$

4. Optimal state sequence backtracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, ..., 1 \tag{3.39}$$

The Viterbi algorithm is similar in its implementation to the forward and backward algorithms (excluding the backtracing step). The key difference is that for the recursion step the Viterbi algorithm maximizes over previous states (as seen in (3.35) as opposed to summing over them (which was shown in (3.20). Step 1 of the Viterbi algorithm initializes the probabilities for δ and ψ when $t = 1$. This step

Figure 6: Trellis diagram illustrating the recursion step of the Viterbi algorithm.



sets $\psi_1(i) = 0$ for all states and finds $\delta_1(i)$ by using the initial state distribution π alongside our observation probability matrix B.

The second step is the recursion step, which is illustrated in Figure 6. The left side of the diagram shows the states at time t from s_1 to s_N, as well as the $\delta_t(i)$ that are taking into consideration all the probabilities from time $t = 1$ to time t. The bold arrow represents the highest scoring probability of all a_{ij} when moving between state s_i at time t to state s_j at time $t + 1$, in this case the most likely transition to state s_j is from s_3. Now the new probability of the highest scoring path until time $t + 1$ is given by $\delta_{t+1}(j) = \delta_t(3)b_j(o_{t+1}$. The previous state that was chosen to have the maximum probability of transitioning to state s_j was state s_3 and so $\psi_{t+1}(j)$ stores the value 3 so that we can find out the optimal state sequence when we backtrack in step 4 of the Viterbi algorithm.

The third step terminates the program and gives the probability of the most likely state sequence as P^*, as well as revealing the most likely final state of this sequence, q_T^*. The fourth step uses the back pointers that were saved during the recursion step to backtrack to time $t = 1$. This backtracking allows us to find the most likely (highest scoring/optimal) state sequence. We should note that, while the Viterbi algorithm allows us to find the optimal state sequence, one of it's weaknesses is that there is no easy way to find the second, third, fourth, etc. best state sequences.

3.6 Training the model

Whilst the solutions to the first and second problem provide valuable information, it is unlikely that these solutions will be useful if the model λ isn't accurate. Adjusting the model parameters so that they best fit the observations (and are therefore more accurate) is our aim when we solve problem 3. As a reminder, the third problem seeks to adjust the model parameters (A, B, π) to maximize the probability of the observation sequence $O = (o_1, o_1, ..., o_t)$ given the model. The fact that we want to adjust these parameters whilst dealing with incomplete data (the hidden states) makes this a difficult problem to solve. Though it's difficult to solve, the fact that the hidden Markov models can be continually and efficiently re-estimated is part of what makes them so useful.

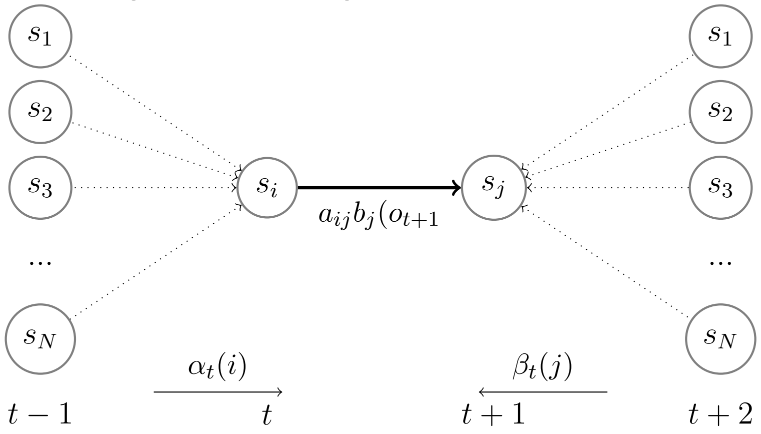
The best available solution is to choose $\lambda = (A, B, \pi)$ such that $P(O|\lambda)$ is locally maximized by using an iterative procedure known as the Baum-Welch algorithm, which was first shown by Baum et al in 1970 [4]. This local maximum is denoted by $\hat{\lambda} = \text{argmax}_\lambda P(O|\lambda)$. The Baum-Welch algorithm can be derived as a particular type of expectation-maximisation (EM) algorithm that is specifically used to train HMM's. Standard EM algorithms rely on iterative methods and are used to find maximum likelihood estimates of parameters for models that depend on unobserved latent variables, but are used for more than just HMMs. It is beyond the scope of this project to delve into the theory behind EM algorithms in general, but further information on the topic can be found in an interesting paper by Dempster et al [7].

The Baum-Welch algorithm re-estimates the parameters for a hidden Markov model λ by using iterative updates to improve it's accuracy. The explanation for this algorithm follows the original 1970 paper by Baum et al [4] as well as Rabiner [17]. In order to describe this procedure we are required to define the variable $\zeta_t(i,j)$, which represents the the probability of being in state i and j at times t and $t+1$ respectively, given the model and the observation sequence O. Therefore, $\zeta_t(i,j)$ is given by:

$$\zeta_t(i,j) = P(q_t = s_i, q_{t+1} = s_j | O, \lambda) \tag{3.40}$$

The equation makes it clear that $\zeta_t(i,j)$ represents the joint event that the model is in state s_i at time t and state s_j at time $t+1$. The sequence of calculations required for the calculation in equation (3.40) are shown in Figure 7 [17]. The left side shows how the forward algorithm has been used to calculate $\alpha_t(i)$, and the backward algorithm has been used to calculate $\beta_{t+1}(j)$. The probability of state s_i transitioning to state s_j from time t to time $t+1$ is shown in the middle. From the illustration in figure 7 and the definitions of the forward and backward variables, we

Figure 7: Trellis diagram illustrating the calculations required to find $\zeta_t(i,j)$.



can now define $\zeta_t(i,j)$ by:

$$\begin{aligned}\zeta_t(i,j) &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} \\ &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}\end{aligned} \tag{3.41}$$

The numerator is just $P(q_t = s_i, q_{t+1} = s_j, O|\lambda)$ and we divide this by $P(O|\lambda)$, in a similar way to the calculation of $\gamma_t(i)$, to normalize $\zeta_t(i,j)$ and therefore make it a probability measure.

Earlier we defined $\gamma_t(i)$ to be the probability of being in state s_i at time t, given the observation sequence O and the model. This $\gamma_t(i)$ can be related to $\zeta_t(i,j)$ by summing over j:

$$\gamma_t(i) = \sum_{j=1}^{N}\zeta_t(i,j) \tag{3.42}$$

We can use $\gamma_t(i)$ to find the expected number of times that a specific state s_i is visited - this is equivalent to the expected number of transitions made from state s_i. This is done by summing over $\gamma_t(i)$ from time t to $T-1$. We exclude the time slot T from the summation because we are trying to find the expected number of transitions made from state s_i, and if the final state at time T is state s_i, we might inaccurately assume that we transition from s_i one more time than the actual expected value. Hence we have the following equation:

$$\sum_{t=1}^{T-1}\gamma_t(i) = \text{expected number of transitions from } s_i \tag{3.43}$$

The expected number of transitions from state s_i to state s_j is found in a similar manner, by summing over the variable $\zeta_t(i,j)$:

$$\sum_{t=1}^{T-1} \zeta_t(i,j) = \text{expected number of transitions from } s_i \text{ to } s_j \tag{3.44}$$

We use these formulas to create our method for re-estimating the parameters of a HMM λ. Given a model $\lambda = (A, B, \pi)$, we can re-estimate A, B, and π. First we re-estimate π by using the following equation:

$$\hat{\pi} = \gamma_1(i) = \text{number of times in state } s_i \text{ at time } t = 1, \quad 1 \le i \le N \tag{3.45}$$

This represents the expected frequency that we are in state s_i at time 1. The new transition probabilities $\hat{a}_{ij}$ are given by:

$$\begin{aligned}\hat{a}_{ij} &= \frac{\text{expected number of transitions from state } s_i \text{ to state } s_j}{\text{expected number of transitions from state } s_i} \\ &= \frac{\sum\limits_{t=1}^{T-1} \zeta_t(i,j)}{\sum\limits_{t=1}^{T-1} \gamma_t(i)}, \quad 1 \le i \le N, \ 1 \le j \le N\end{aligned} \tag{3.46}$$

Finally, the re-estimated values for the observation array $\hat{b}_j(k)$ are given by:

$$\begin{aligned}\hat{b}_j(k) &= \frac{\text{expected number of times in state } s_j \text{ and observing symbol} v_k}{\text{expected number of times in state } s_j} \\ &= \frac{\sum\limits_{\substack{t=1 \\ \text{s.t.} o_t = v_k}}^{T} \gamma_t(j)}{\sum\limits_{t=1}^{T} \gamma_t(j)}, \quad 1 \le j \le N, \ 1 \le k \le M\end{aligned} \tag{3.47}$$

The ratio in equation 3.47 is the probability of observing emission k, given that the model is in state s_j, which is the desired value of $b_j(k)$. We use our current model λ to calculate the left-hand sides of equations (3.45), (3.46) and (3.47). The updated values that we find are then used to define our re-estimated model, which is given by $\hat{\lambda} = (\hat{A}, \hat{B}, \hat{\pi})$. It has been proven by Baum and his collaborators [2] [3] that this new model $\widehat{lambda}$ is either just as accurate as the original model λ, or more accurate. If it is equally accurate, then the initial model λ defines a critical point of the likelihood function, and we have that $\hat{\lambda} = \lambda$. If our new model is more accurate, we have that $P(O|\hat{\lambda}) > P(O|\lambda)$, i.e., we have discovered a new model $\widehat{lambda}$ that is more likely to have produced the observation sequence O.

Using the equations above, we can repeat the re-estimation calculation using $\hat{\lambda}$ in place of λ. If we see that $P(O|\hat{\lambda})$ improves, we can repeat the process again and again until some limiting point is reached. The final result of this re-estimating method is called a maximum likelihood estimate of the HMM, and once this point is reached the HMM can no longer be trained.

It is clear that re-estimation is therefore an iterative process. The first step is to initialize our starting model $\lambda = (A, B, \pi)$ with our best available guess. If we have such little data that we can't create a reasonable estimate, we can choose random values such that $\pi_i \approx \frac{1}{N}$, $a_{ij} \approx \frac{1}{N}$ and $b_j(k) \approx \frac{1}{M}$ [20]. It's crucial that A, B and π are randomized, since exactly uniform values will result in a local maximum from which the model cannot climb.

An important feature of this re-estimation process is that the stochastic constraints of the HMM parameters are automatically satisfied for each iteration. That is, $\hat{\pi}$, $\hat{A}$ and $\hat{B}$ are all row stochastic.

To conclude our discussion of the solution to problem 3, we summarise the procedure for training a model λ in four steps:

1. Initialize our first estimate of the model, $\lambda = (A, B, \pi)$.
2. Compute the necessary variables, $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i)$ and $\zeta_t(i, j)$.
3. Re-estimate the model as $\hat{\lambda} = (\hat{A}, \hat{B}, \hat{\pi})$ by using equations (3.45), (3.46) and (3.47).
4. If $P(O|\hat{\lambda}) > P(O|\lambda)$, replace λ with $\hat{\lambda}$ and repeat the process from step 2.

3.7 Continuous Emission Hidden Markov Models

All of the theory until this moment has assumed that we are working with discrete emissions that are taken from a finite observation alphabet V. Discrete observations allow us to use a discrete probability density within each state of the model. In practice, many applications of HMM's deal with continuous observations, the most common example being speech analysis. In the fourth section of this project we will treat stock market prices as a continuous observation that can be used to analyse the underlying hidden states, so an understanding of continuous observations is crucial. The following explanation for continuous observations is given in more detail in Rabiner's paper [17].

For continuous emission Hidden Markov Models, the observation probabilities are modelled using Gaussian Mixture Models (GMMs). A Guassian mixture model is a probabilistic model that assumes that all of the the data points are generated from a mixture of a finite number of Gaussian (Normal) distributions with unknown parameters μ and Σ. The continuous observation density is given by a restricted probability density function (pdf), and is represented by a finite mixture of the form [17]:

$$b_j(\overrightarrow{O}) = \sum_{m=1}^{M} c_{jm}\eta(\overrightarrow{O}, \overrightarrow{\mu_{jm}}, \Sigma_{jm}) \quad 1 \leq j \leq N \tag{3.48}$$

We have that $\overrightarrow{O}$ is the observation vector that's being modeled and c_{jm} is the mixture coefficient (the weight) of the m-th mixture in state j. The variable $\overrightarrow{\mu_{jm}}$ represents the mean vector for the m-th mixture component in state j and Σ_{jm} gives the covariance matrix for the m-th mixture component, again in state j. Combining these variables, $\eta(\overrightarrow{O}, \overrightarrow{\mu_{jm}}, \Sigma_{jm})$ is the probability of observing the observation vector from the multi-dimensional Gaussian distribution. The mixture gains c_{jm} satisfy the stochastic constraint:

$$\begin{aligned} \sum_{m=1}^{M} c_{jm} &= 1, \quad 1 \leq j \leq N, \\ c_{jm} &\geq 0, \quad 1 \leq j \leq N, \ \ 1 \leq m \leq M \end{aligned} \tag{3.49}$$

The pdf given in 3.48 is therefore properly normalized, i.e.:

$$\int_{-\infty}^{\infty} b_j(x)dx = 1, \quad 1 \leq j \leq N \tag{3.50}$$

This probability density function can be used to approximate, arbitrarily closely, any continuous density function that is finite. Therefore, we can apply this equation to a wide range of different problems, including the stock market. The next section explains how we can model a real-world continuous observation HMM that takes its continuous observations from stock market, using Python.

4 Analysis of the stock market using Hidden Markov Models

The stock market is a network which provides a platform for almost all major economic transactions in the world at a dynamic rate, known as the stock value, which is based on market equilibrium [11]. This network contains large amounts of historical data that is easily accessible for any individual that wants to invest in a specific company. This rich source of data in the form of time-series based data-sets make the stock market a prime candidate for the experimental usage of Hidden Markov Models, for several reasons.

First of all, HMMs are specifically designed to be used to analyze and predict time series based data, using the algorithms laid out in previous sections. Secondly, they have the ability to construct a prediction model that can determine the unobserved underlying states behind a process, given that the only available information is a visible set of observations. When analyzing the stock market we have no knowledge of the hidden states that govern it, we only know the current and historical stock prices (our observations). The transitions between these hidden states are based on information that is not easily accesible to an investor - changes in the economy, in-house company policy (mergers etc.), new products, business scandals and many other factors. Finally, HMMs use training algorithms (Baum-Welch) to make a given model λ more accurate. This improvement in accuracy is largely dependant on how much available training data there is. The stock market contains decades of potential training data, and so provides a suitable environment for creating highly accurate HMMs.

Though HMMs have primarily been used within the fields of bioinformatics, speech recognition and handwriting recognition, there are a small number of papers that have applied HMMs to the stock market. Weigend and Shi [22] were the first to achieve significant results using HMMs to predict the stock market. They predicted changes in the trajectories of different financial time series data, and found that HMMs can produce better predictions than autoregressive models [22]. A more recent paper by Hassan and Nath [12] compared Hidden Markov Models to artificial neural networks (ANN), which had already been established as a method for successfully predicting time series behaviour, and found that HMMs were just as effective. Their success at predicting the stock market was judged by their relative mean absolute percentage errors (MAPE), which is also what I will use to analyse my own stock market predictions. The most recent paper that used HMMs for stock market prediction was written in 2012 by Dhingra and Gupta [11]. They used MATLAB to write a program that trains a Gaussian emission HMM, using several months of training data, and then uses the newfound model to predict the daily close price of stocks. Similar to Hassan [12], they found that HMMs produced a lower MAPE than ANNs and also found that they were more accurate than auto-regressive moving average models (ARMA).

The approach taken in this project is similar to the one taken by Dhingra and Gupta [11], but the code behind it is written in Python, not MATLAB. They downloaded data on the daily high, low, opening and closing stock prices and used this data to calculate the fractional changes on a daily basis. The fractional changes are then stored in a vector and represent the continuous observations. This set of observations is used to train a continuous Hidden Markov Model which can then go on to predict future observations. None of the papers outlined above included their code so I have partially written my own program in Python by adapting pre-existing code on HMMs that I found on GitHub [10] and the open source Python package hmmlearn [13].

4.1 Method

Assume that we have a continuous emission Hidden Markov Model that we are using to model the stock price data as a time series. Our HMM is denoted by λ, and is given by $\lambda = (A, B, \pi)$, following the notation outlined in section 3. Since the observations are continuous, the observation probabilities are modelled as Gaussian Mixture Models, as shown in equation (3.48).

The HMM λ is trained using the Baum-Welch algorithm [4]. This is done using the fit() method included in the Python package hmmlearn, more of which can be read about by looking at the documentation [13].

There are limited features available when analysing daily stock data, namely the opening price, the closing price, the highest price the stock reached and the lowest price. We seek to compute the closing stock price for a day, given the opening stock price for that day, and some previous d days data. For our model, the observations are discovered by finding the fractional changes in the daily stock data and producing the 4-dimensional observation vector o_t. The equations for the fractional changes are given by:

$$\begin{aligned} frac_{change} &= \frac{(close - open)}{open} \\ frac_{high} &= \frac{(high - open)}{open} \\ frac_{low} &= \frac{(open - low)}{open} \end{aligned} \tag{4.1}$$

Using these equations, we can now define our 4-dimensional observation vector o_t as:

$$o_t = (frac_{change}, frac_{high}, frac_{low}) \tag{4.2}$$

Fractional changes are used instead of the actual values for open, close, etc. because stock prices can have large amounts of variation over a long time period (multiple years), whereas fractional changes remain more constant. When training the model we will use several years worth of data so fractional changes will provide more accurate data.

After we have trained the model, it is tested using an approximate Maximum a Posteriori (MAP) approach, which follows the theory outlined by Dhingra and Gupta [11]. A MAP estimate is an estimate of an unknown quantity, that equals the mode of the posterior distribution. We assume that there is a latency of d days while forecasting future stock values - the value we select for d can also be seen as the order of our Hidden Markov Model. If $d = 5$, we could forecast tomorrows stock given the previous 5 days of data.

We can reframe our problem as follows - given the Hidden Markov Model λ and the stock values for d days $(o_1, o_2, o_3, ..., o_d)$ as well as the opening stock value for the $(d+1)$-st day, we want to compute the closing stock value for the $(d+1)$-st day. Since our observation vector is composed of fractional changes and not the actual stock prices we compute the closing stock value for the $(d+1)$-st day by estimating the fractional change for that day. We calculate the closing price by rearranging the equation for $frac_{change}$ using simple algebra:

$$\textit{close} = \textit{open} \times (1 + frac_{\textit{change}}) \tag{4.3}$$

To predict the fractional change for the $(d+1)$-st day, we need to compute the MAP estimate of the observation vector o_{d+1}. Let $\bar{o}_{d+1}$ represent the MAP estimate of the observation on the $(d+1)$-st day, given the values of the previous d days. Then, as Gupta and Dhingra have shown [11], we have:

$$\begin{aligned} \bar{o}_{d+1} &= \underset{o_{d+1}}{\operatorname{argmax}} P(o_{d+1}|o_1, o_2, ..., o_d, \lambda) \\ &= \underset{o_{d+1}}{\operatorname{argmax}} \frac{P(o_1, o_2, ..., o_d, o_{d+1}, \lambda)}{P(o_1, o_2, ..., o_d, \lambda)} \end{aligned} \tag{4.4}$$

The observation vector o_{d+1} is varied over all possible values. Due to the fact that the denominator in (4.4) is constant with respect to o_{d+1}, we can rewrite our MAP estimate as:

$$\bar{o}_{d+1} = \underset{o_{d+1}}{\operatorname{argmax}} P(o_1, o_2, ..., o_d, o_{d+1}|\lambda) \tag{4.5}$$

The joint probability $P(o_1, o_2, ..., o_{+1}|\lambda)$ is calculated by using the forward-backward algorithm that was outlined in Definition 3.1. If we assume that $frac_{change}$, and all the other features of our observation vector, are continuous variables it is very difficult to optimize $\bar{o}_{d+1}$. To deal with this problem we instead compute the probability over a discrete set of possible values of o_{d+1}, which are shown in Table 2. We then find the set of fractional changes $o_t = (frac_{change}, frac_{high}, frac_{low})$ that maximizes the posterior probability $P(o_1, ..., o_{d+1}|\lambda)$, hence the name MAP HMM.

Table 2: Range of values for MAP estimation

Observation	**Minimum Value**	**Maximum Value**	**Number of points**
$frac_{change}$	-0.1	0.1	50
$frac_{high}$	0	0.1	10
$frac_{low}$	0	0.1	10

The computational complexity of the forward-backward algorithm for this experiment can be given by big O notation as $O(N^2 d)$ [11], where d is the latency and N is the total number of states in the model. The forward-backward algorithm is repeated over the discrete set of possible values of o_{d+1} shown in Table 2. For this experiment, we set $N = 4$, since the observation vector is 4-dimensional, and $d = 10$, as this amount of latency has been shown to minimize the error in the results [12] [11]. Table 2 shows that we have $50 \times 10 \times 10 = 5000$ possible values of o_{d+1}, so the procedure is repeated 5000 times.

4.2 Implementation

The open-source Python package hmmlearn is used to implement various HMM functions (training the model using Baum-Welch, forward backward algorithm) and to initialize the starting probabilities π and the transition probabilities A [13]. Execution of the experiment as a whole can be describe in the following five steps:

1. *Choose HMM Parameters:* The parameters for our HMM λ are given by the following:

   - Dimension of the observations, (o_t), are given by (D), $D = 3$
   - Our HMM is ergodic, meaning that any state can reach any other given state
   - Latency (how many previous days of data is taken into consideration) (d), $d = 10$
   - Percentage of data that is used to train the model (R), $R = 67\%$
   - Percentage of data that is tested (T), $T = 33\%$
   - Number of unobserved Hidden states (N), $N = 4$

   Both [12] and [11] suggest that four unobserved states are ideal because the data dimensionality is also four. The HMM is ergodic because we know so little about the hidden states that there is no reason to restrict any transitions. Through experimentation using the code, I found that using approximately $\frac{2}{3}$ of the data to train the model produced the most accurate results, hence $R = 67\%$.

2. *Initialization:* When initializing the model parameters $\lambda = (A, B, \pi)$, the start probabilities π and the transition probabilities A are assumed to be uniform across all states, and are initialized by the Python package hmmlearn [13]. Financial data is downloaded from Yahoo finance using the Pandas datareader package, and the fractional changes that form the observation vector o_t (which are given in equation 4.1) are calculated. Once the observation array has been created, the data is split up into a training data-set and a test data-set. The training data (the set of observations covering 77% of the time) is used to calculate the parameters for the observations matrix B, including the variables μ and Σ which represent the parameters of the multinomial Gaussian distribution. These parameters are estimated by using the fit() method on the GaussianHMM class which is provided within the hmmlearn package [13]. Now that we have a trained model, we can attempt to predict the daily close prices for the other 33% of the data, and see how accurate they are when compared to the daily close prices in the test data-set.

3. *Prediction:* To compute the MAP estimate $\bar{o}_{d+1}$, we compute the probabilty values over a range of possible values of the tuple

$$o_t = (frac_{change}, frac_{high}, frac_{low})$$

and find the maximum. The range of values is listed in Table 2. The range of values considered for the $frac_{change}$ observations is larger because $frac_{change}$ is what is ultimately used for the stock prediction, so a greater level of precision is required.

4. *Results:* The performance of the program is evaluated by the metric Mean Absolute Percentage Error (MAPE). MAPE is the average absolute error between the predicted stock values and the actual stock values, in percentage form, and is given by the following equation [11]:

$$\text{MAPE} = \frac{1}{t}\sum_{i=1}^{t}\frac{|p_i - a_i|}{|a_i|} \times 100\% \tag{4.6}$$

Where p_i represents the predicted stock value on day i, a_i represents the actual stock value, and t is the number of days for which the data is tested. Once the predictions have been made in step 3, the data predicted and actual data is stored in an excel document and then the MAPE is calculated.

5. *Visualization:* Once all the results have been recorded and the MAPE has been found, the predicted stock close prices and the actual stock close prices can be plotted against each-other in a graph. This is done using the matplotlib package.

4.3 Results

The program was tested on four different stocks, Tesla, Apple, Google and Amazon. The code used to create the program is shown in Appendix A. For Tesla, Apple and Google, the same time period was covered, with one year of total data being downloaded from the 1st of January 2019 until the 1st of January 2020. I noticed that when the algorithm analyzed Google, which has a far higher stock value than both Apple and Tesla, the predictions for the closing stock price were less accurate. Amazon has a similarly high stock price, so I increased the total amount of historical data used to see if the accuracy would be increased. The details of the training and testing periods are shown in Table 3.

Table 3: Training and Test Data-sets

Observation	Training Data		Test Data	
	Start Date	End Date	Start Date	End Date
Apple Inc.	01/01/2019	02/09/2019	03/09/2019	01/01/2020
Tesla Inc.	01/01/2019	02/09/2019	03/09/2019	01/01/2020
Alphabet Inc.	01/01/2019	02/09/2019	03/09/2019	01/01/2020
Amazon.com Inc.	01/01/2017	06/01/2019	07/01/2019	01/01/2020

The stock market is only open on weekdays, and therefore excludes weekends and bank holidays. Because of this, even though we've chosen the testing data to be 33% of the total data recorded, the testing data actually analysed for Google, Apple and Tesla covers a period of 84 days, instead of roughly 121. Similarly for Amazon, the amount of days covered by the testing period is 249 days as opposed to the expected 359.

Table 4: Mean Absolute Percentage Error for the program

Stock Name	MAP HMM Model
AAPL	0.735087
TSLA	1.488147
GOOG	2.918786
AMZN	0.827285

The MAPE values found are displayed in Table 4. As a whole, all the values of MAPE are less than 3%, and if we exclude Google they are all less than 1.5%. When MAPE is used for statistical analysis, any value below 10% is viewed as excellent [9]. The fact that the MAPE values are so low suggests that the program was highly successful at predicting the close prices for the $(d+1)$-st day, given some previous d days data and the open price for the $(d+1)$-st day. Another key takeaway is that there is still potential for the model to be improved further - after seeing that an expensive stock (Google is currently valued at $1434.87) resulted in the greatest MAPE value, I increased the amount of training data used to predict an even more expensive stock (Amazon is currently valued at $2471.04), and this resulted in Amazon having the second lowest MAPE. This would imply that the models precision can be improved if the quantity of data used to create the observations is increased.

A direct comparison of the actual stock values and the predicted values can be seen graphically for each of the stocks , over the next four pages, in Figures 8, 9, 10, and 11. The code that produced the graphs can be seen in Appendix B.

Figure 8: Actual Close Price vs Predicted Close Price for Apple from 03/09/2019 to 01/01/2020

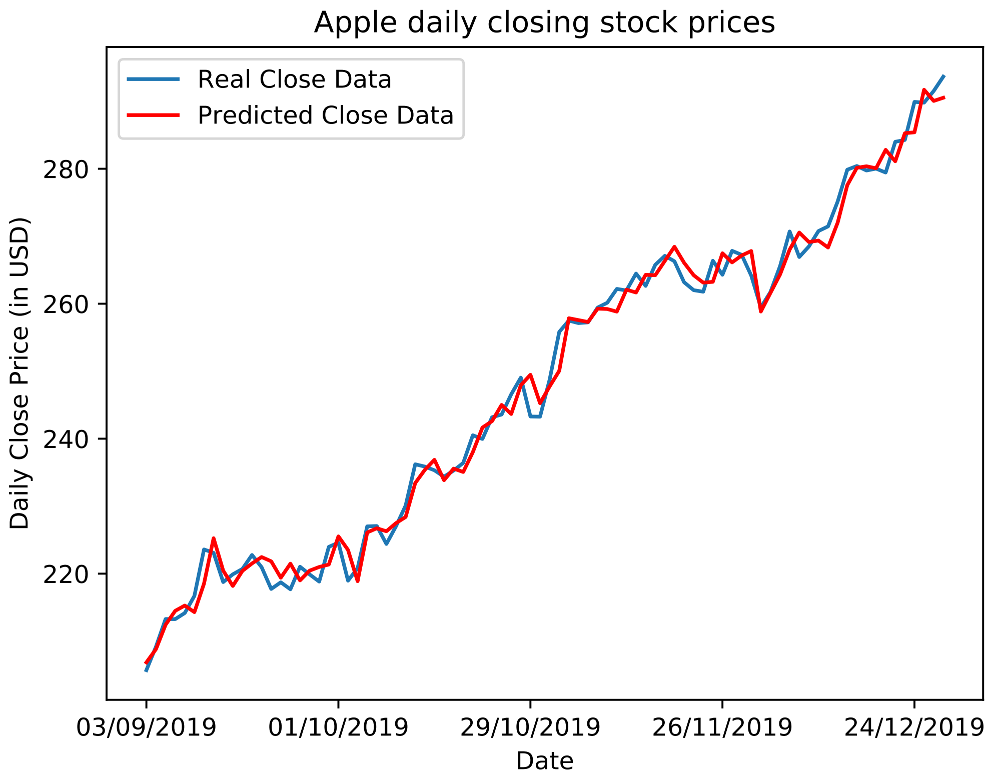


Figure 9: Actual Close Price vs Predicted Close Price for Tesla from 03/09/2019 to 01/01/2020

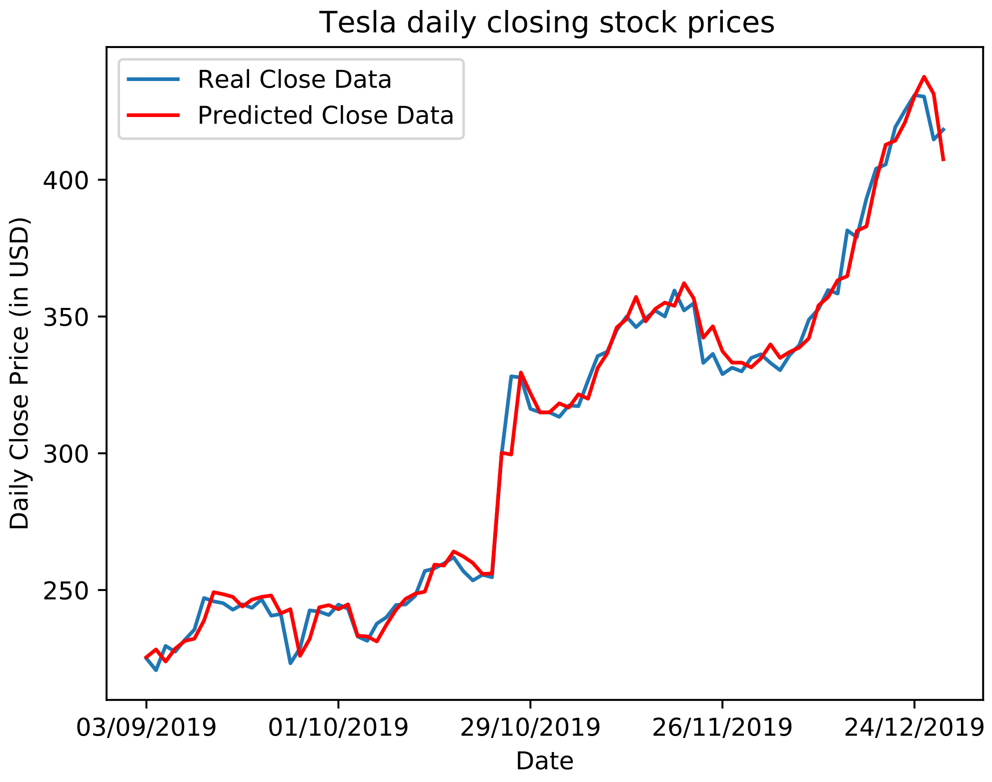


Figure 10: Actual Close Price vs Predicted Close Price for Google from 03/09/2019 to 01/01/2020

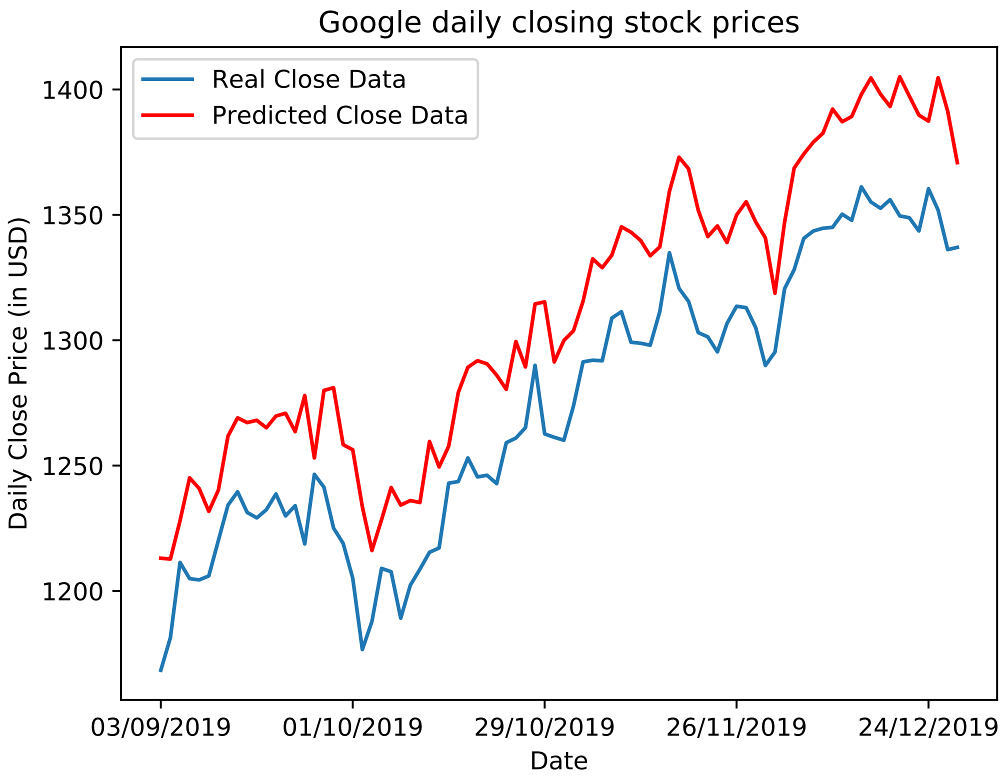
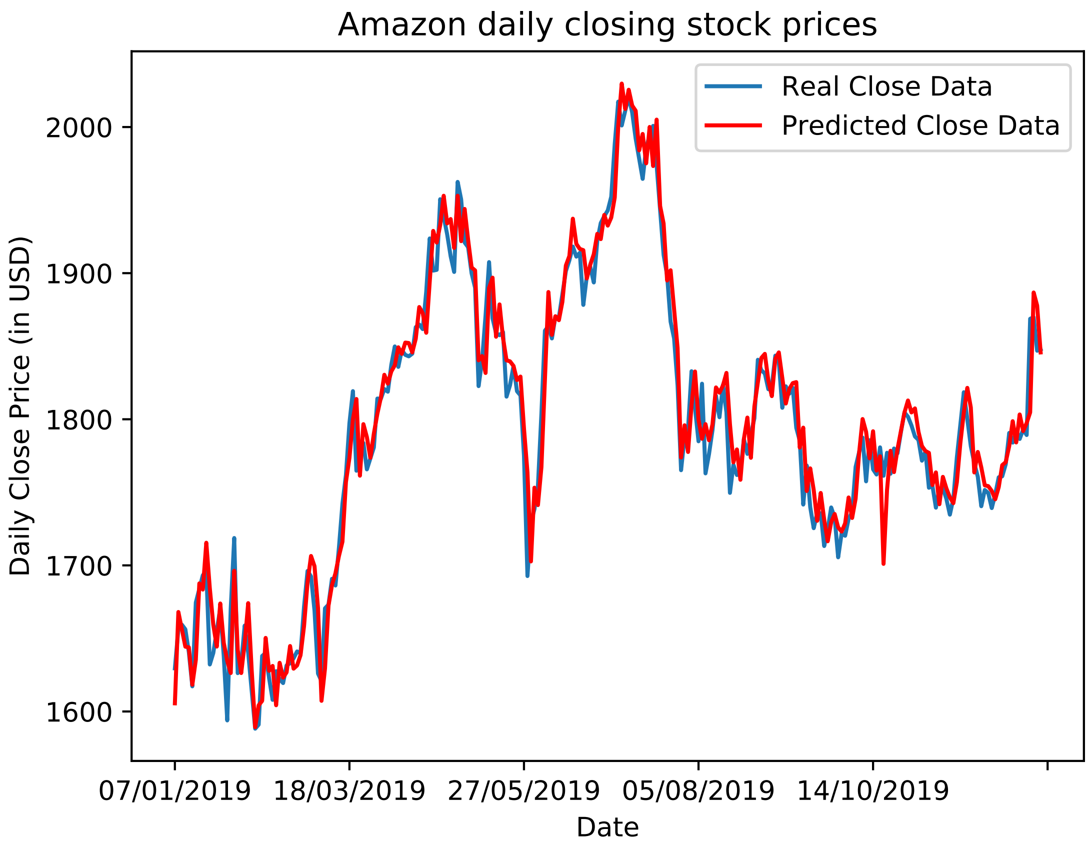


Figure 11: Actual Close Price vs Predicted Close Price for Amazon from 07/01/2019 to 01/01/2020



5 Summary and conclusions

This project has sought to present an in depth explanation of the theory behind Hidden Markov Models, beginning with simple concepts such as discrete Markov chains, and extending this theory to more sophisticated models such as continuous emission HMMs. This research is then utilized to perform an experiment analysing the stock market. It is structured in such a way that a reader with little prior experience with HMMs will be able to understand how they work and then either go on to either research HMMs further, or start to apply them within their specific area of interest. The project laid out the notation most commonly used for HMMs and then went on to work through a simple example to elucidate the key differences between standard Markov processes and Hidden Markov Models. After that, the three main problems that HMMs seek to solve (in other words, the three most common reasons to use HMMs) were explained, and their solutions, the forward-backward, Viterbi, and Baum-Welch algorithms, were outlined in detail. The explanation for each of these algorithms was accompanied by relevant diagrams to make some of the more abstract concepts more understandable. Finally, a brief detour is made to explain how continuous emission HMMs work, as they're a prerequisite for the experiment performed in section 4.

The experiment presents a HMM based MAP estimator for stock prediction. The model assumes that there are four underlying states which emit the set of visible observations given by o_t. A program (given in Appendix A) was written in Python to download stock market data for a given time period, split it into training and test data, train the HMM using GaussianHMM (from the hmmlearn package), and then predict the close prices and compare the predicted values with the actual values from the test data-set.

The program found that using a MAP-HMM model is a viable method for stock market prediction, since the MAPE values across all four stocks were below 3%. These MAPE values are shown in Table 4. This result confirms similar findings by Gupta, Hassan and Weigend [11] [12] [22]. It is interesting to note that the accuracy of the prediction can be improved by increasing the amount of training data used, as seen in the results for Amazon.

The current approach has several assumptions that could be tested in future work. The first of these is the assumption that each stock is independent of the other stocks. In reality, stocks from companies that operate within similar domains (cars, finance, technology, etc.) will likely follow similar trends and thus may be dependent on each other. Another assumption is that the model would be just as accurate during a time for which the global economy is in a recession. All of the data taken in the experiment was taken during a boom, with all four stock prices going up over the time period. If the data had been trained during a boom, but then been tested at a time when global markets were struggling, it may have been less accurate. To prepare for this, future work may seek to train a model using data that includes a period of global recession. Future work may also attempt to achieve further performance improvements by increasing the total amount of data used to train the model in general, regardless of economic recession or boom, as this project did not consider more than four years of data for any of the stocks.

6 Bibliography

References

[1] L.E. Baum, and T. Petrie, Statistical inference for probabilistic functions of finite state Markov Chains, *The Annals of Mathematical Statistics*, (1966), 1554–1562, Available at: `https://projecteuclid.org/DPubS/Repository/1.0/Disseminate?handle=euclid.aoms/1177699147&view=body&content-type=pdf_1`

[2] L.E. Baum, and J.A. Eagon, An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology, *Bull. Amer. Math. Soc.*, **73(3)**, (1967), 360–363, Available at: `https://projecteuclid.org/download/pdf_1/euclid.bams/1183528841`

[3] L.E. Baum, and G.R. Sell, Growth transformations for functions on manifolds, *Pacific J. Math*, **27(2)**, (1968), 211–227, Available at: `https://projecteuclid.org/download/pdf_1/euclid.pjm/1102983899`

[4] L.E. Baum, T. Petrie, G. Soules, and N. Weiss, A Maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains, *The Annals of Mathematical Statistics*, **41(1)**, (1970), 164–171, Available at: `https://projecteuclid.org/download/pdf_1/euclid.aoms/1177697196`

[5] P. Blunsom, Hidden Markov Models, *Citeseer*, (2004), 1-7, Available at: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.1016&rep=rep1&type=pdf`

[6] E. Cinlar, *Introduction to Stochastic Processes*, Prentice-Hall, 1974. Print.

[7] A. Dempster, N. Laird, and D.B. Rubin, Maximum Likelihood from incomplete data via the EM algorithm, *Journal of the Royal statistical Society*, **39**, (1977), 1–38, Available at: `https://www.jstor.org/stable/2984875?read-now=1&seq=1#page_scan_tab_contents`

[8] G.D. Forney, The Viterbi algorithm, *Proceedings of the IEEE*, **61(3)**, (1973), 268-278, Available at: `https://ieeexplore.ieee.org/document/1450960`

[9] M. Gilliland, Forecasting FAQs, *The Business Forecasting Deal: Exposing Myths, Eliminating Bad Practices, Providing Practical Solutions* , SAS Institute, (2010), 193–246, Available at: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119199885.app1`

[10] GitHub - analyse_data.py, Code by NMZivkovic that uses HMMs in Python, Available at: `https://gist.github.com/NMZivkovic/a04d813557ae31d0890036162fb68eb8#file-analyse_data-py`

[11] A. Gupta and B. Dhingra, Stock Market Prediction Using Hidden Markov Models, *Students Conference on Engineering and Systems*, (2012), 1–4, Available at: `https://ieeexplore.ieee.org/document/6199099`

[12] M.R. Hassan and B. Nath, Stock Market Forecasting using Hidden Markov Model: A New Approach, *5th International Conference on Intelligent Systems Design and Applications*, (2005), 192–196, Available at: `https://ieeexplore.ieee.org/abstract/document/1578783?casa_token=ZOJ1iHH_OBwAAAAA:yjFclldsgMkb4d9AHrh14R4GM3td60Ls30_0-DPHaeyU-qZwMobdBZEaZe2nkzDyoJrQOhC2`

[13] hmmlearn, Documentation for the Python package hmmlearn, Available at: `https://hmmlearn.readthedocs.io/en/latest/index.html`

[14] N. Li and M. Stephens, Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data, *Genetics*, **165(4)**, (2003), 2213–2233, Available at: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1462870/`

[15] A.A. Markov, An Example of Statistical Investigation of the Text Eugene Onegin Concerning the Connection of Samples in Chains, *Science in Context*, **19(4)**, (2006), 591–600, Available at: `http://www.alpha60.de/research/markov/DavidLink_AnExampleOfStatistical_MarkovTrans_2007.pdf`

[16] A.B. Poritz, Hidden Markov Models: A guided tour, *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, **1**, (1988), 7–13, Available at: `https://ieeexplore.ieee.org/document/196495`

[17] L.R. Rabiner, A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, *Proceedings of the IEEE*, **77(2)**, (1989), 257–285, Available at: `https://www.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/tutorial%20on%20hmm%20and%20applications.pdf`

[18] S. Ross, *A First Course in Probability 8th Edition*,Pearson Education, Cambridge, 2010. Print.

[19] R. Serfozo, *Basics of Applied stochastic processes*, Springer, (2009), 1–9, Available at: `http://www.stat.yale.edu/~jtc5/251/readings/Basics%20of%20Applied%20Stochastic%20Processes_Serfozo.pdf`

[20] M. Stamp, A revealing introduction to Hidden Markov Models, *Department of Computer Science San Jose State University* (2004), 26–56, Available at: `http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf`

[21] A. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Transactions on Information Theory*, **13(2)**,

(1967), 260–269, Available at: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1054010`

[22] A.S. Weigend and S. Shi, Predicting Daily Probability Distributions of S&P500 Returns, *Journal of Forecasting*, **19(4)**, (2000), 375–392, Available at: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1054010`

A Python code for the experiment

The following code is partially my own creation and partially adapted from code on HMMs that can be found on GitHub [10]. Run the program stock_analysis.py –company AAPL in command prompt and replace AAPL with any desired stock name. The dates can also be changed, as long as they are of the form "year-day-month".

```
"""
Usage: stock_analysis.py --company=<company>
"""
import warnings
from tqdm import tqdm
import pandas as pd
import logging
from pandas_datareader import data
import itertools
from sklearn.model_selection import train_test_split
import numpy as np
from hmmlearn.hmm import GaussianHMM
from docopt import docopt
import warnings


args = docopt(doc=__doc__, argv=None, help=True,
              version=None, options_first=False)

#Type in python stock_analysis.py --company AAPL, and replace
#AAPL with your desired stock in your command prompt
# Supress warning in hmmlearn
warnings.filterwarnings("ignore")

class StockPrediction(object):
    def __init__(self, company, test_size=0.33,
                 n_hidden_states=4, n_latency_days=10,
                 n_intervals_fractional_change=50,
                 n_intervals_fractional_high=10,
                 n_intervals_fractional_low=10):
        self._init_logger()
        self.company = company
        self.n_latency_days = n_latency_days

        self.hmm = GaussianHMM(n_components=n_hidden_states)
        self._split_train_test_data(test_size)

        self._compute_all_possible_outcomes(
            n_intervals_fractional_change,
            n_intervals_fractional_high,
            n_intervals_fractional_low)

    def _init_logger(self):
        self._logger = logging.getLogger(__name__)
        handler = logging.StreamHandler()
        formatter = logging.Formatter(
            '%(asctime)s␣%(name)-12s␣%(levelname)-8s␣%(message)s')
        handler.setFormatter(formatter)
        self._logger.addHandler(handler)
        self._logger.setLevel(logging.DEBUG)


    def _split_train_test_data(self, test_size):
        start_date = '2019-01-01' #change the dates here as
        #desired
        end_date = '2020-01-01'
        # We use the module pandas datareader
        # to download data from Yahoo finance
        used_data = data.DataReader(
            self.company, 'yahoo', start_date, end_date)
        _training_data, testing_data = train_test_split(
            used_data, test_size=test_size, shuffle=False)

        self._training_data = _training_data
        self._testing_data = testing_data
        print(testing_data)
    @staticmethod
    def _extract_observations(data):
        daily_open_price = np.array(data['Open'])
        daily_close_price = np.array(data['Close'])
        daily_high_price = np.array(data['High'])
        daily_low_price = np.array(data['Low'])

        # We compute the fractional change in high,
        # low, and close prices
        # to use as our set of observations
        fractional_change = (daily_close_price - daily_open_price)\
            / daily_open_price
        fractional_high = (daily_high_price - daily_open_price)\
            / daily_open_price
        fractional_low = (daily_open_price - daily_low_price)\
            / daily_open_price

        return np.column_stack((fractional_change,
         fractional_high, fractional_low))

    def fit(self):
        #extract the observations from the training data
        observations = StockPrediction._extract_observations(
            self._training_data)
        #Fit the HMM using the observations
        # from the training data
        # and the hmmlearn method .fit
        self.hmm.fit(observations)

    def _compute_all_possible_outcomes(self,
            n_intervals_fractional_change,
            n_intervals_fractional_high,
            n_intervals_fractional_low):
        fractional_change_range = np.linspace(
            -0.1, 0.1, n_intervals_fractional_change)
        fractional_high_range = np.linspace(
            0, 0.1, n_intervals_fractional_high)
        fractional_low_range = np.linspace(
            0, 0.1, n_intervals_fractional_low)

        self._possible_outcomes = np.array(list(
            itertools.product(
            fractional_change_range,
             fractional_high_range,
              fractional_low_range)))

    def _get_most_likely_outcome(self, day_index):
        previous_data_start_index = max(
            0, day_index - self.n_latency_days)
        previous_data_end_index = max(
            0, day_index - 1)
        previous_data = self._testing_data.iloc[
            previous_data_end_index: previous_data_start_index]
        previous_data_observations= StockPrediction._extract_observations(
            previous_data)
        outcome_score = []
        for possible_outcome in self._possible_outcomes:
            total_data = np.row_stack(
                (previous_data_observations, possible_outcome))
            outcome_score.append(self.hmm.score(total_data))
        most_likely_outcome = self._possible_outcomes[np.argmax(
            outcome_score)]

        return most_likely_outcome

    def predict_daily_close_prices(self, day_index):
        daily_open_price = self._testing_data.iloc[
            day_index]['Open']
        predicted_fractional_change, _, _ = self._get_most_likely_outcome(
            day_index)
        return daily_open_price * (1 + predicted_fractional_change)

    def predict_close_prices_for_n_days(self, days):
        predicted_close_prices = []
        for day_index in tqdm(range(days)):
            predicted_close_prices.append(
                self.predict_daily_close_prices(
                    day_index))
        return predicted_close_prices

    def actual_daily_close_prices(self):
        actual_close_prices = self._testing_data['Close']
        return actual_close_prices


stock_predictor = StockPrediction(company=args['--company'])
stock_predictor.fit()
#Select the amount of days you would like to predict
#Here it is 84 because 84 days of data is tested
#For the one year timeframe
predicted_close = stock_predictor.predict_close_prices_for_n_days(84)
real_close_values = stock_predictor.actual_daily_close_prices()
predicted_close_values = pd.DataFrame(predicted_close)
#Put the recorded data into excel documents for analysis
predicted_close_values.to_excel(
    r'YourData␣\stockpredicted.xlsx', index = False)
real_close_values.to_excel(
    r'YourData\stockactual.xlsx', index = False)
```

B Python code for plotting graphs

The following code uses the matplotlib library to plot the actual and predicted close prices.

```
import pandas as pd
import matplotlib.pyplot as plt
#Import the predicted and actual close values
#Which are stored in an excel document
fileName = 'companydata.csv'
df = pd.read_csv(fileName)

ax = plt.gca()

df.plot(kind='line',x='Date',y='Real␣Close␣Data',ax=ax)
df.plot(kind='line',x='Date',y='Predicted␣Close␣Data',
    color='red', ax=ax)
plt.ylabel('Daily␣Close␣Price␣(in␣USD)')
plt.title('Company␣daily␣closing␣stock␣prices')
#Save the pdf to your PC
plt.savefig(r'YourData\companystockpredicted.xlsx')
plt.show()
```