Stock market forecasting using continuous hidden markov models

Aadam Ul Haq, Lyris Xu, Yuanhe Zhang

Summer 2022

1 Introduction

Within this research project, we examine the ability of hidden Markov models (HMMs) in forecasting stock market states and patterns. A Markov process is a statistical model describing sequential data with the assumption that the probability of each state's occurrence is only dependent on the previous state. Hidden Markov models incorporate the aspect of hidden states (unobserved states which influence each observable state).

Some very popular models are Machine learning models such as autoregressive integrated moving average (ARIMA) or artificial neural networks (ANNs), as well as stochastic models[1]. Our contribution is valuable as there is less material available about stochastic methods, particularly Hidden Markov Models. This is because most stochastic methods are geometric Brownian motion models. There have been some studies that show HMMs perform better than long short term models (LSTMs)[2].

2 What is a Hidden Markov Model?

A hidden Markov model is a doubly stochastic process, where one set of stochastic processes is hidden and only observed through the other set of stochastic processes [3]. Before defining them formally, we will consider an example, adapted from[4].

Consider a scenario in which there are two temperatures, hot and cold. Suppose we have evidence that a hot day is followed by a hot day with probability 0.8, and a cold day followed directly by a cold day is with probability 0.6. We can summarise these events with the following matrix:

$$\begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{pmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{pmatrix} \end{array}$$

H is hot, and C is cold.

This is a stochastic matrix as all the entries are non-negative and the sums of the rows are 1. Now consider the probabilities of having ice-cream depending on the temperature. If it is a hot day, the probability of having ice-cream is 0.9. If it is cold, the probability of having ice-cream is 0.2. We can summarise these in the following matrix:

$$\begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{cc} I & N \end{array} \\ \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix} \end{array}$$

I means ice-cream is eaten, and N means no ice-cream is eaten. This matrix is also a stochastic matrix.

There is a clear relationship between eating ice-cream and the temperature. In this example, the state is the daily temperature, in which the transitions between states is a Markov process. If we consider historical data in which the temperature is unknown, these states are hidden, however are able to be determined by looking at ice-cream sales for example.

Our state/transition matrix is the first matrix

$$A = \begin{pmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{pmatrix}$$

Our observation/emission matrix is the second matrix

$$B = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix}$$

Suppose we also had an initial distribution

$$\pi = \begin{pmatrix} 0.7 & 0.3 \end{pmatrix}$$

We could then ask some questions, for example, given a person ate ice-cream over a three day period in the sequence $\mathcal{O} = 0, 1, 1$, where 0 represents not eating ice-cream and 1 representing they did, what is the most likely sequence of Markov process given this data. HMMs are needed to answer questions like this.

More formally, the definitions are that for a Markov chain, where n is a positive integer and $i_0, i_1, \ldots, i_n$ are possible events, we have

$$\mathbb{P}\left(X_n = i_n \mid |X_{n-1} = i_{n-1}\right) = \mathbb{P}\left(X_n = i_n \mid |X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}\right).$$

Equivalently, this can be expressed as the probability of the next step is only determined by the previous step. A Markov chain can simply be written as 3-tuple $\theta = (S, M, \pi)$ where S is the set of spaces e.g $\{\text{Hot }, \text{Cold}\}$, M is the transition matrix such as Matrix A above, and π is simply our initial distribution vector.

More formally, the prior definition described is a first order time homogenous Markov chain. A hidden Markov model builds upon these concepts. The states S are unseen and are not directly observed at a time. So $s(t)$ is unknown at a time t. Every point however emits a signal, v, that can be observed with a definite probability. This can be seen as another stochastic process and we can see $v(t)$ at time t. The probability of the emission at time t is dependent on $s(t)$ so is the conditional probability $\mathbb{P}\left(v(t) \mid |s(t)\right)$. Let Σ be the possible states of the observed set e.g $\{\text{Eat Ice-cream}, \text{Does not eat Ice-cream}\}$, and let δ be the emission matrix. Then we can describe a hidden Markov model as the 5-tuple $\theta = (S, M, \Sigma, \delta, \pi)$.[5]

There are several algorithms that are helpful for discovering natures of HMMs. One of the most popular is known as the forward algorithm. The next section goes into more detail into the algorithm used.

3 Methodology and Building the Model

3.1 General Set-up

We propose a continuous hidden markov model for stock prediction. We model the stock data as time series dataset. First, we denote an HMM

$$\lambda = (\pi, A, B) \tag{1}$$

Q: State space

$$Q = (q_1, q_2, ..., q_N) \tag{2}$$

I: State sequence

$$I = (i_1, i_2, ..., i_T) \tag{3}$$

O: Observation space

$$O = (o_1, o_2, ..., o_N) \tag{4}$$

O: Observation sequence

$$O = (o_1, o_2, ..., o_T) \tag{5}$$

π: initial distribution

$$\pi = (\pi_1, \pi_2, ..., \pi_N) \tag{6}$$

A: transition matrix

$$A = [a_{ij}]_{N \times N} \tag{7}$$

$$a_{ij} = P(i_{t+1} = q_j \mid i_t = q_i) \tag{8}$$

B: emission matrix

$$A = [b_j(k)]_{N \times N} \tag{9}$$

$$b_j(k) = P(o_t = v_k \mid i_t = q_j) \tag{10}$$

In order to fit for our research problem, for a continuous hidden markov model, the emission probability need to be modelled as a Gaussian Mixture Models (GMMs):

$$b_j(O_t) = \sum_{m=1}^{M} c_{jm} N(O_t, \mu_{jm}, \Sigma_{jm}) \tag{11}$$

where:

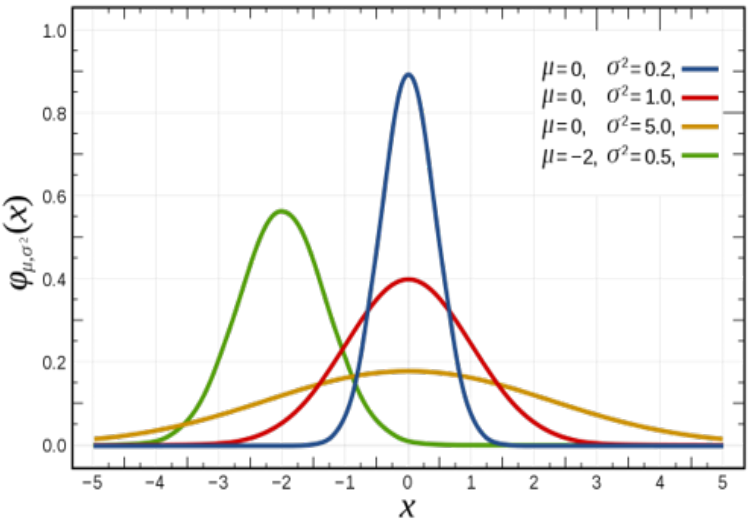


Figure 1: Gaussian Mixture Models: This is an example of GMMs. We can clearly see that how weights behave in the model to make seperate univariate gaussian distribution components to be a mixture model.

- M is the number of Gaussian Mixture components
- c_{jm} is the weight of the m^{th} mixture component in the state j.
- μ_{jm} is the mean vector for the m^{th} component in the j^{th} state.
- $N(O_t, \mu_{jm}, \Sigma_{jm})$ is the probability of observing O_t in the multivariate Gaussian distribution.

In our model, we have a three-dimensional vector which represents three observation states:

$$O_t = (\frac{close - open}{open}, \frac{high - open}{open}, \frac{open - low}{open}) = (fracChange, fracHigh, fracLow) \quad (12)$$

where open is the day opening value, close is the day closing value, high is the day high, and low is the day low. We use fractional changes along to model the variation in stock data which remains constant over the years.

3.2 Training process

Given the HMM model $\lambda = (\pi, A, B)$ and the stock values for d days which we can get our observation sequence $(O_1, ..., O_d)$. Then we can assume a latency of d days while forecasting the stock. Now if given the open price of $(d+1)_{th}$ day, we aim to compute the close price for $(d+1)_{th}$ day. Alternately, this is equivalent to estimate the fractional change $\frac{close-open}{open}$ for the $(d+1)_{th}$ day.

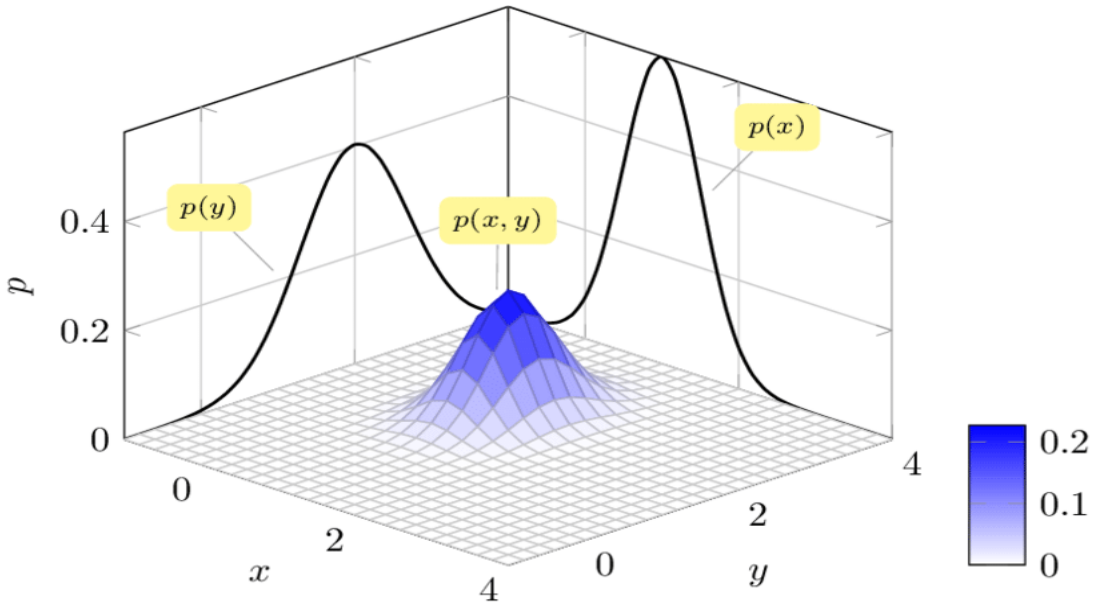


Figure 2: Multivariate Gaussian Distribution: Take a bivariate gaussian distribution as an example of multivariate gaussian distribution. In a two-dimensional space, there are two univariate gaussian distribution in each dimension. And if we concentrate them together in this 2-dimensional space, then we can get a bivariate gaussian distribution which is a kind of MGDs.

Next, we need to introduce a definition called Maximum a Posterior (MAP). First, the posterior probability is a type of conditional probability which update the prior probability from information summarized by the likelihood (Data!). A prior probability represents one's belief before this person see some evidence. The logic of MAP is that the optimal parameter should be to maximize the posterior probability. MAP can be used for unobserved point estimation through data.

Now we define $\hat{O}_{d+1}$ be the MAP of the observation on the $(d+1)_{th}$. Given the observation states sequence $(O_1, O_2, ..., O_d)$ for the first d days and the HMM model $\lambda = (A, B, \pi)$, then we can estimate the MAP $\hat{O}_{d+1}$:

$$\hat{O}_{d+1} = argmax_{O_{d+1}} P(O_{d+1}|O_1, O_2, ..., O_d, \lambda) \tag{13}$$

According to Bayes's Rule and we already have the information that given HMM model λ the observation states sequence $(O_1, O_2, ..., O_d)$ s.t. $P(O_1, O_2, ..., O_d, \lambda) = 1$,

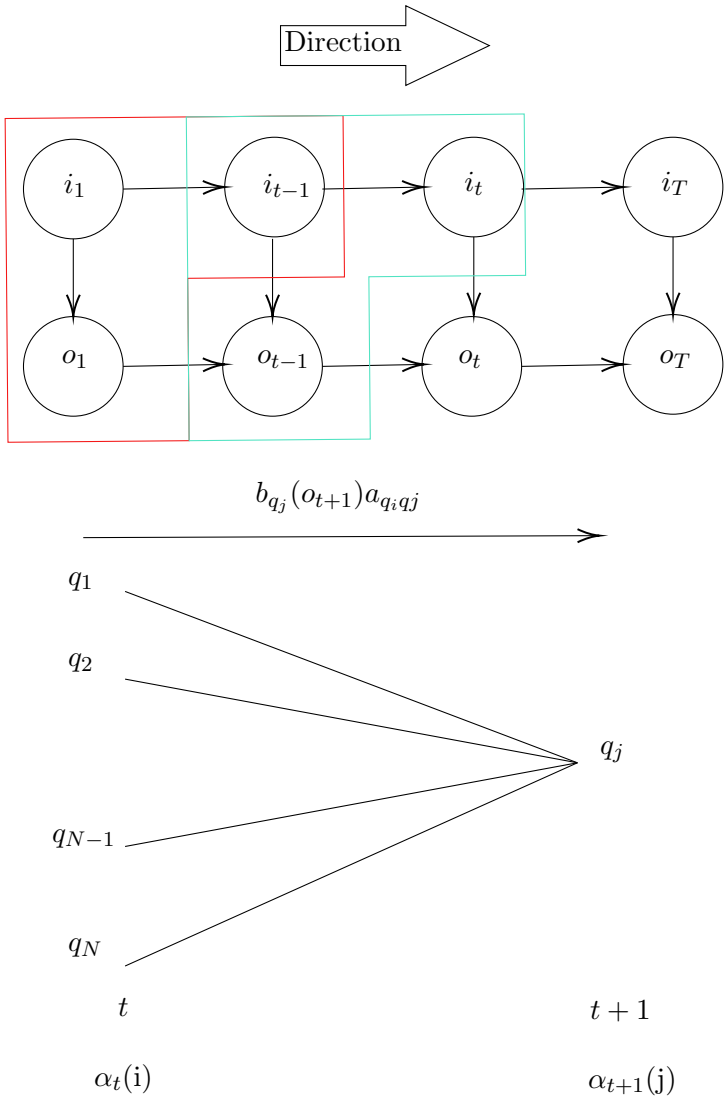
$$\hat{O}_{d+1} = argmax_{O_{d+1}} \frac{P(O_{d+1}, O_1, O_2, ..., O_d|\lambda)}{P(O_1, O_2, ..., O_d, \lambda)} = argmax_{O_{d+1}} P(O_{d+1}, O_1, O_2, ..., O_d|\lambda) \tag{14}$$

For this probability, we can evaluate it using the Forward-Backward algorithm for HMMs.

Algorithm 1 Forward algorithm

Require: Observation sequence O and HMM λ

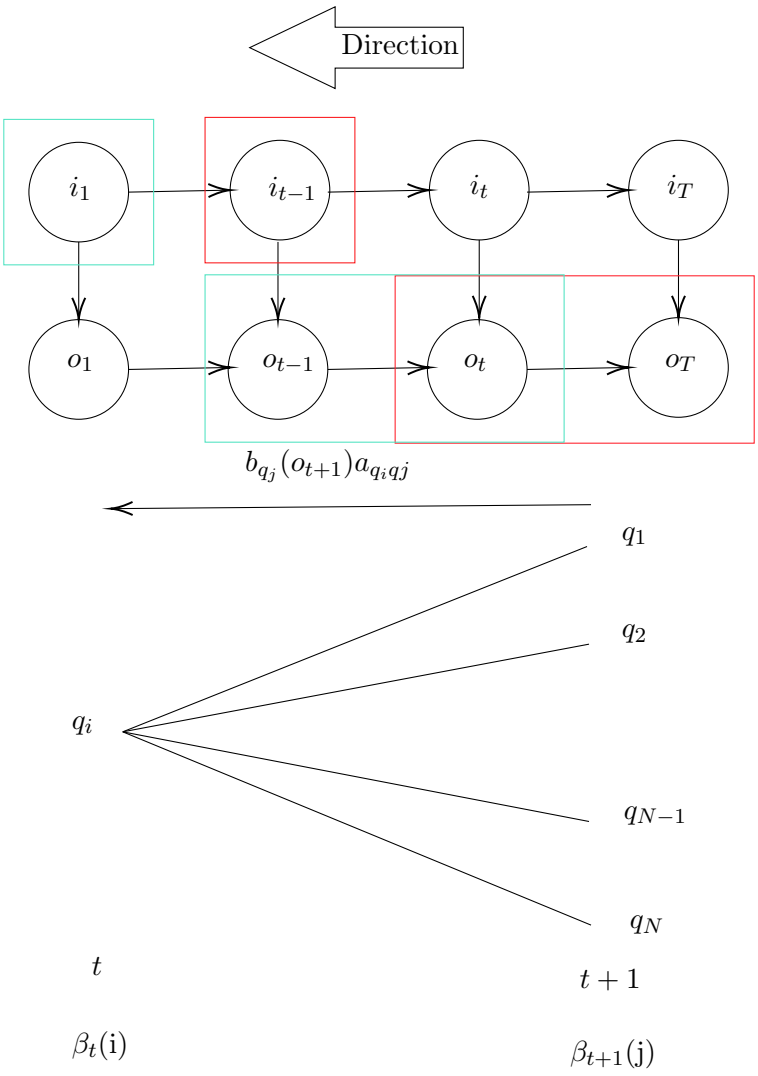
$t \leftarrow 1$
for $i = 1, ..., N$ **do**
  $\alpha_1(i) \leftarrow \pi_{q_i} b_{q_i}(o_1)$
end for
while $t < T$ **do**
  for $j = 1, ..., N$ **do**
    $\alpha_{t+1}(j) = \sum_{i=1}^{N} b_{q_j}(o_{t+1}) a_{q_i q_j} \alpha_t(i)$
  end for
end while
$P(O \mid \lambda) \leftarrow \sum_{i=1}^{N} \alpha_T(i)$



Algorithm 2 Backward algorithm

Require: Observation sequence O and HMM λ

$t \leftarrow T$
for $i = 1, ..., N$ **do**
  $\beta_1(i) \leftarrow 1$
end for
while $t > 0$ **do**
  for $i = 1, ..., N$ **do**
    $\beta_t(i) \leftarrow \sum_{j=1}^{N} a_{q_i q_j} b_{q_j}(o_{t+1}) \beta_{t+1}(j)$
  end for
  $t \leftarrow t - 1$
end while
$P(O \mid \lambda) \leftarrow \sum_{i=1}^{N} \pi_{q_i} b_{q_i}(o_1) \beta_1(i)$



Finally, the close price for $(d+1)_{th}$ day can be computed by using the open price of

$(d+1)_{th}$ day and $\frac{close-open}{open}$ from MAP $\hat{O}_{d+1}$,

$$predicted_close_price = \hat{open} * \hat{fracChange} + \hat{open} \tag{15}$$

where the hat represented values from MAP.

3.3 Elements of Implementation (with Code)

We use the open source hmmlearn package in Python. And the American Airlines stock dataset is from Kaggle.

3.3.1 HMM parameters set-up:

- Number of Hidden States n = 4
- Number of mixture components for each state m = 5
- Dimension of observations states D = 3
- Latency d = 10 days

3.3.2 Initialization

We used k-means methods to initialize the model parameters $\lambda = (A, B, \pi)$ which followed by methods from [6].

3.3.3 Prediction

To compute the MAP estimate $\hat{O}_{d+1}$, we compute the probability values over a range of possible values of the tuple $(\frac{close-open}{open}, \frac{high-open}{open}, \frac{open-low}{open})$ and find the maximum.

3.3.4 Code

```
import logging
import itertools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from hmmlearn.hmm import GaussianHMM
from sklearn.model_selection import train_test_split
from tqdm import tqdm

class StockPredictor():
    def __init__(self, test_size=0.33,
                 n_hidden_states=4, n_latency_days=10,
                 n_steps_frac_change=50, n_steps_frac_high=10,
                 n_steps_frac_low=10):
        self._init_logger()

        self.n_latency_days = n_latency_days

        self.hmm = GaussianHMM(n_components=n_hidden_states)
```

```
20
21          self._split_train_test_data(test_size)
22
23          self._compute_all_possible_outcomes(
24              n_steps_frac_change, n_steps_frac_high, n_steps_frac_low)
25
26      def _init_logger(self):
27          self._logger = logging.getLogger(__name__)
28          handler = logging.StreamHandler()
29          formatter = logging.Formatter(
30              '%(asctime)s %(name)-12s %(levelname)-8s %(message)s')
31          handler.setFormatter(formatter)
32          self._logger.addHandler(handler)
33          self._logger.setLevel(logging.DEBUG)
34
35      def _split_train_test_data(self, test_size):
36          data = pd.read_csv('C:/360Downloads/PythonProject/aa.us.txt')
37          _train_data, test_data = train_test_split(
38              data, test_size=test_size, shuffle=False)
39
40          self._train_data = _train_data
41          self._test_data = test_data
42
43      @staticmethod
44      def _extract_features(data):
45          open_price = np.array(data['Open'])
46          close_price = np.array(data['Close'])
47          high_price = np.array(data['High'])
48          low_price = np.array(data['Low'])
49
50          # Feature engineering
51          frac_change = (close_price - open_price) / open_price
52          frac_high = (high_price - open_price) / open_price
53          frac_low = (open_price - low_price) / open_price
54
55          return np.column_stack((frac_change, frac_high, frac_low))
56
57      def fit(self):
58          self._logger.info('>>> Extracting Features')
59          feature_vector = StockPredictor._extract_features(self._train_data)
60          self._logger.info('Features extraction Completed <<<')
61
62          self.hmm.fit(feature_vector)
63
64      def _compute_all_possible_outcomes(self, n_steps_frac_change,
65                                         n_steps_frac_high, n_steps_frac_low):
66          #Compute the probability over a range of possible values of the tuple
    (fracChange, fracHigh, fracLow)
67          frac_change_range = np.linspace(-0.000001, 0.000001,
    n_steps_frac_change)
68          frac_high_range = np.linspace(0, 0.000001, n_steps_frac_high)
69          frac_low_range = np.linspace(0, 0.000001, n_steps_frac_low)
70
71          self._possible_outcomes = np.array(list(itertools.product(
72              frac_change_range, frac_high_range, frac_low_range)))
73
```

```
    def _get_most_probable_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._test_data.iloc[previous_data_end_index:
    previous_data_start_index]
        previous_data_features = StockPredictor._extract_features(
            previous_data)

        outcome_score = []
        for possible_outcome in self._possible_outcomes:
            total_data = np.row_stack(
                (previous_data_features, possible_outcome))
            outcome_score.append(self.hmm.score(total_data))
        most_probable_outcome = self._possible_outcomes[np.argmax(
            outcome_score)]

        return most_probable_outcome

    def predict_close_price(self, day_index):
        open_price = self._test_data.iloc[day_index]['Open']
        predicted_frac_change, _, _ = self._get_most_probable_outcome(
            day_index)
        #Predict by the open price and the MAP value
        return open_price * (1 + predicted_frac_change)

    def predict_close_prices_for_days(self, days, with_plot=False):
        predicted_close_prices = []
        for day_index in tqdm(range(days)):
            predicted_close_prices.append(self.predict_close_price(day_index))
        #Add plots to visualise the results
        if with_plot:
            test_data = self._test_data[0: days]
            days = np.array(test_data['Date'], dtype="datetime64[ms]")
            actual_close_prices = test_data['Close']

            fig = plt.figure()

            axes = fig.add_subplot(111)
            axes.plot(days, actual_close_prices,label="actual")
            axes.plot(days, predicted_close_prices,label="predicted")
            axes.set_title('American Airlines Stock Prediction')

            fig.autofmt_xdate()

            plt.legend()
            plt.show()

            return predicted_close_prices, actual_close_prices


stock_predictor = StockPredictor()
stock_predictor.fit()
p = stock_predictor.predict_close_prices_for_days(100, with_plot=True)

#Testing metric: Mean Square Error
p = np.array(p)
```

Trials	fracChange	fracHigh	fracLow	MSE
1	(-1,1)	(0,1)	(0,1)	4.6930996244939625
2	(-0.1,0.1)	(0,0.1)	(0,1)	1.9340589314296153
3	(-0.01,0.01)	(0,0.01)	(0,0.01)	1.6446590966086576
4	(-0.001,0.001)	(0,0.001)	(0,0.001)	1.6359331971606512

Table 1: Mean Square Error under different range of values for MAP estimation

```
129
130 predicted_close_prices = np.array(p[0])
131 actual_close_prices = np.array(p[1])
132
133 m = predicted_close_prices - actual_close_prices
134 s = m*m
135 MSE = (1/len(predicted_close_prices))*np.sum(s)
136
137 print(MSE)
```

3.3.5 Results

Run the code above and we can get the 500-days predicted results showing in the Figure 3 (under Trial 2 parameters set-up in Table 1.). Also, we set up a testing metric for evaluating the accuracy of our models called Mean Square Error (MSE). MSE is defined as the average of the error squares. It is also known as the metric that evaluates the quality of a forecasting model or predictor. MSE also takes into account variance (the difference between anticipated values) and bias (the distance of predicted value from its true value). The closer MSE is to zero, the better.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{16}$$

where n is the total number of values in the test set, y_i is the i_{th} actual value, $\hat{y}_i$ is the i_{th} predicted value.

As we can see in the Table 1, the MSE decrease when we decrease the range of values for MAP estimations. But when the range value is further reduced, the change in MSE is not obvious.

4 Conclusion and Discussion

The purpose of this research was to explore the effectiveness of a hidden Markov model in predicting the patterns of the stock market. Using the hmmlearn package, a model was trained upon specified parameters and stock market data to predict stock market values to a high degree of accuracy.

The task of stock market forecasting involved a process of feature engineering. Given opening, closing, high and low values of each day of the stock market. We needed to decide

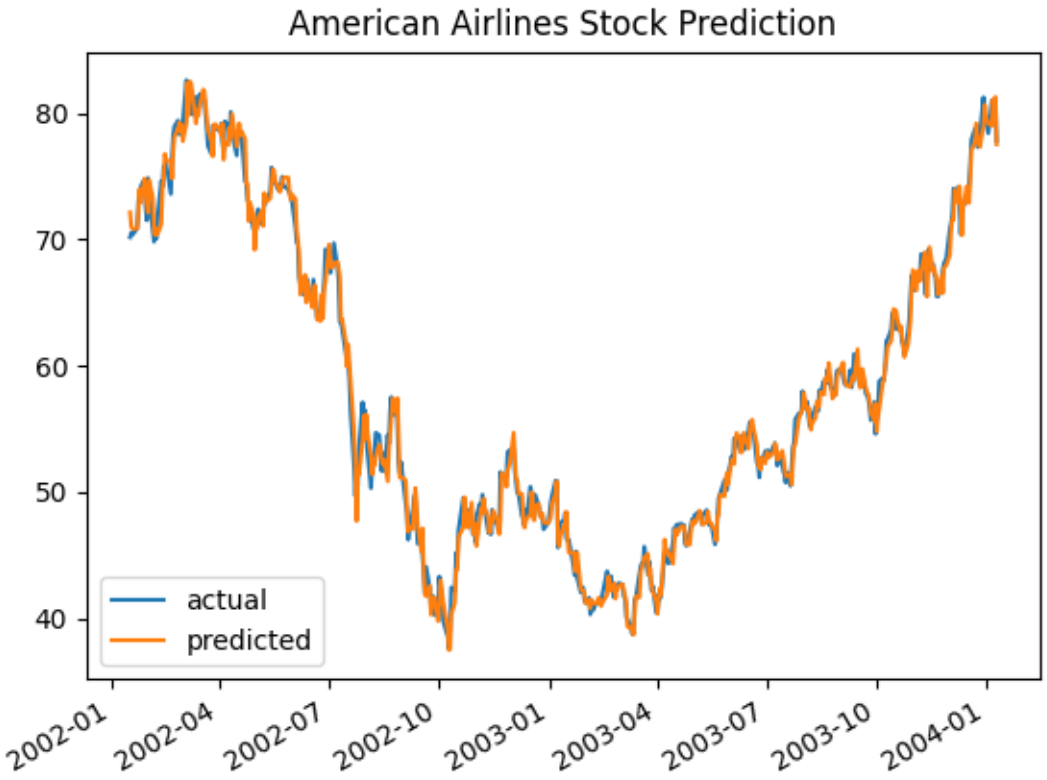


Figure 3: Results for American Airlines Prediction

what new variables to create to serve as observable or hidden states, based on existing models and our understanding of the stock market.

Within this project, we used a vector of fractional change, fractional high and fractional low as observed states. By using fractional variables, we did not have to account for inflation over the series of stock market values. However, there are existing HMMs for stock market prediction which have used other variables for observation states such as two states of "increasing" and "decreasing" with hidden states as discrete intervals from the raw closing values of the stock market. Further exploration of this project may thus be conducted by training the model on different observation and hidden states of stock market data.

A challenge to stock market data is the continuous characteristic of stock market values which may be discretized through rounding or intervalization. However, accuracy is lost when such a set of continuous values is modeled as discrete states. Thus, this directs us to some of the limitations of using this model for stock market predictions.

The performance of hidden Markov models for stock forecasting data may be measured up against alternative models. HMMs assume the Markov property where the probability of each consecutive state is conditional only on the current state. As in:

$$\mathbb{P}\left(X_{t+1} = s \mid |X_0 = x_0, X_1 = x_1, \ldots, X_t = x_t\right) = \mathbb{P}\left(X_{t+1} = s \mid |X_t = x_t\right) \tag{17}$$

Each X is a random variable taking a value within state space S.

Thus, HMMs may be considered to have a "memory-less" approach to modelling sequential data. This is in contrast to models with long term memory such as recurrent neural networks (RNNs) which retains memory of what it has already processes in training. Models such as LTSMs (long short-term memory) and GRU may be explored in modelling stock market returns.

However, it is essential to keep in mind the difficulty of forecasting stock market prices due the unpredictability of many factors (political, economical, etc.) and the inherent "randomness" in such a system.

References

[1] Islam M R, Nguyen N. Comparison of Financial Models for Stock Price Prediction. In *Journal of Risk and Financial Management*. vol. 13, pp. 1, 2020.

[2] de Wit B. Stock prediction using a Hidden Markov Model versus a Long Short-Term Memory. In *Doctoral Dissertation*, pp. 9-13, 2019.

[3] Rabiner L, Juang B. An introduction to hidden Markov models. In *IEEE ASSP magazine*, vol. 3(1), pp. 3, 1986.

[4] Name. A revealing introduction to hidden Markov models In *Department of Computer Science San Jose State University*, pp. 26-29, 2004.

[5] Kohlschein C. An introduction to hidden Markov models. In *Probability and Randomization in Computer Science. Seminar in winter semester*, vol. 2007, pp. 1-4, 2006.

[6] Gupta A, Dhingra B. Stock market prediction using hidden markov models. In *2012 Students Conference on Engineering and Systems. IEEE*, vol. 2012, pp. 1-4.