Adrovic Belmin
Di Cino Luca
Proença Raphael

4th Semester

Experimental Mathematics: Stock trading with hidden Markov models

Project supervisor: George Kerchev

Table of contents

Stock trading with hidden Markov models

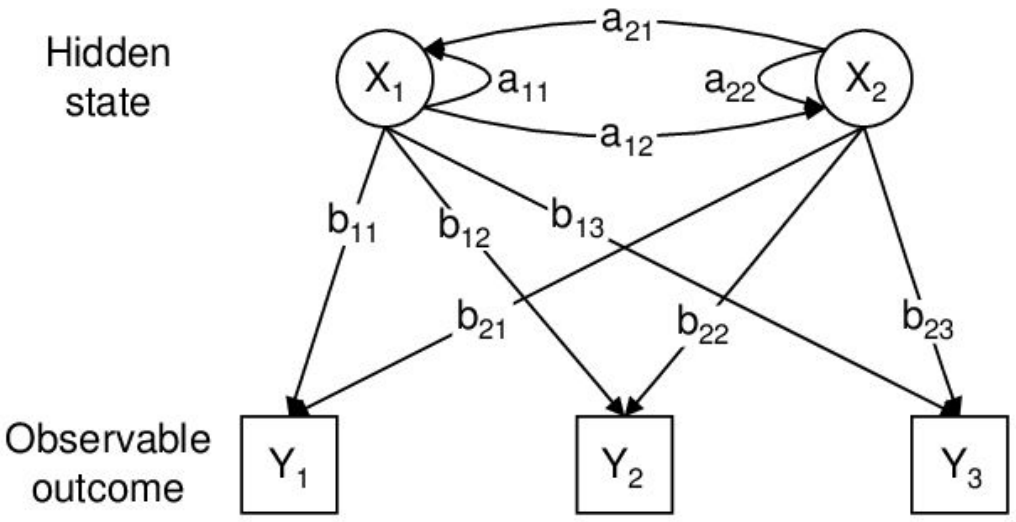
Introduction

The aggregation of buyers and sellers of stocks also called shares is called a stock market. A stock or a share represents ownership claims on businesses. Stock trading consists of buying shares low and selling them high by profiting in doing so. The goal of our project was to be able to predict economic regimes and stock prices using the ***Hidden Markov Model*** (HMM), a probabilistic model. After gathering economic data from the company ***Electronic Arts***, we tried to model it with a HMM. In order to predict the stock prices, we needed to train the model while optimizing the model parameters.

Part I: Hidden Markov Model

Hidden Markov Model

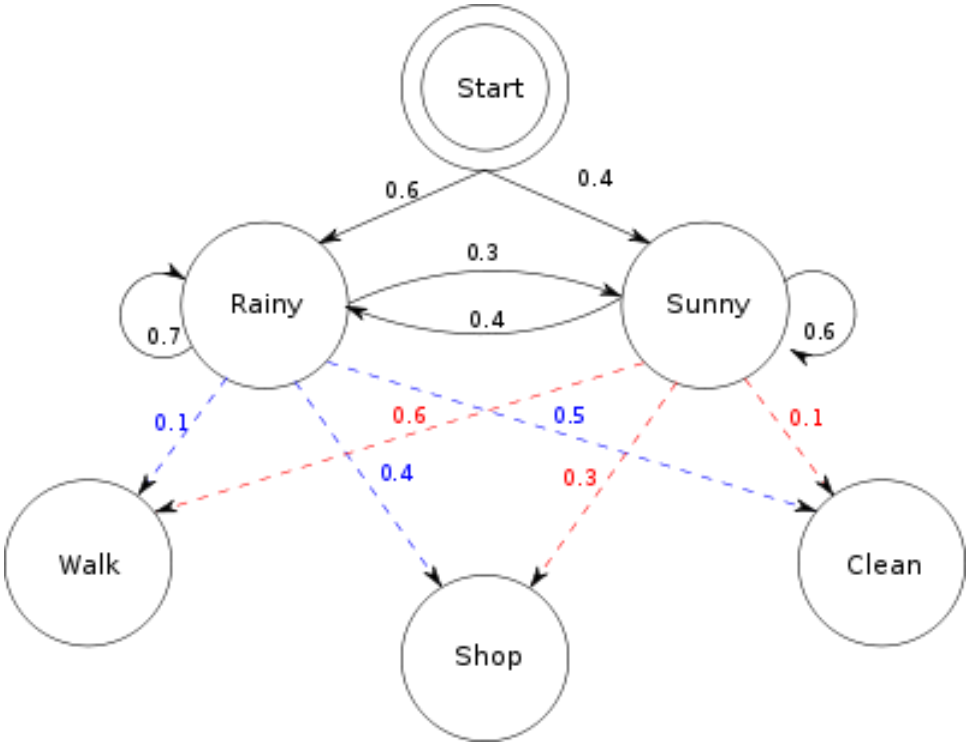
Named after the russian mathematician Andrey Andreyevich, the Hidden Markov Models is a doubly stochastic process where one of the underlying stochastic process is hidden. The hidden process is a Markov chain going from one state to another but cannot be observed directly. The other process is observable and depends on the hidden states. The goal of HMM is to capture the hidden information from the observables. HMM is widely used in speech recognition and in forecasting of the stock market.



X - hidden states
Y - observables
a - **transition probabilities**: probability of moving from one state to another state
b - **emission probabilities**: probability of an observation being generated from a state
X_0 - **initial probability distribution**: probability that the Markov chain will start at a certain state

Example:

An application of the HMM is often the determination of the weather which is hidden. The hidden states in our example are sunny and rainy. The activities of a certain person which depend on the weather can be observed. The observables are the following activities: walk, shop or clean. The transition probabilities here represent the change of weather from one day to another. The emission probabilities represent the likelihood of an activity being performed on each day depending on the state of the weather.



Let's suppose that for two consecutive days this person went for a walk. We will furthermore assume that it rained the first day and the second day it was sunny.

P((walk,walk),(rainy,sunny)) = P((walk,walk)|(rainy,sunny)) P(rainy,sunny)
= P(walk|rainy) P(walk|sunny) P(sunny|rainy) P(rainy)
= 0.1 * 0.6 * 0.3 * 0.6 = 0.0108

Hidden Markov Model for Stock trading

HMM are capable of predicting and analyzing time-based phenomena, hence, they are very useful for financial market prediction. The price of the stock, in this case our observable, is impacted by hidden volatility regimes. The hidden states are namely low volatility regimes, high volatility regimes and neutral volatility regimes. The goal here is to estimate the most likely regime, including the associated time varying means and volatilities, in order to create a reliable predictive model. So by analyzing the stock behavioral pattern in the past, we can forecast its future outcomes.

Three basic problems of HMMs

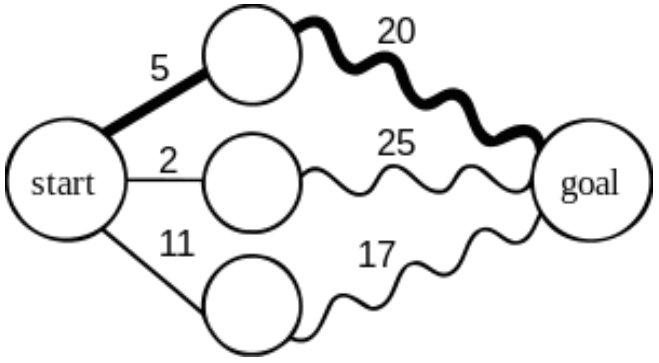
- Evaluation: computation of the probability of the observation sequence
  → Forward-Backward algorithm

- Learning: determination of the parameters of the model
  → Baum-Welch algorithm

- Decoding: determination of the most probable state sequence
  → Viterbi algorithm

Forward-Backward algorithm

The Forward-Backward algorithm computes the posterior marginals of all hidden state variables. This means that it gives the probability distribution with relevant evidence and each background being taken into account. To achieve this it makes use of the principle of dynamic programming, which means simplifying a complicated problem by breaking it down into simpler sub-problems, as we can see on this image.



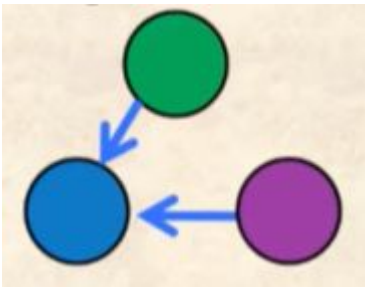
The algorithm uses two passes, the first pass goes forward in time and the second pass goes backward. The first pass computes a set of forward probabilities which provide the probability of ending up in any particular state. In the second pass, the algorithm computes a set of backward probabilities which provide the probability of observing the remaining observations given any starting point. So we have two sets of probabilities that can then be combined to obtain the distribution over states at any specific point in time given.

Conclusion:

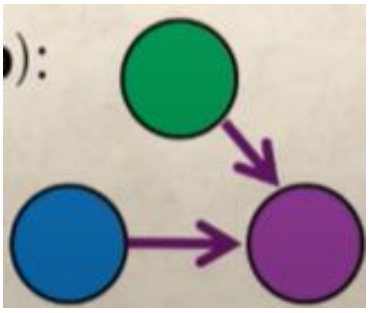
The Forward-Backward algorithm is used to find the most likely state for any point in time.

Baum-Welch algorithm

The Baum-Welch algorithm is used to find the unknown parameters of a hidden Markov model. It's a special case of the EM algorithm (expectation–maximization algorithm) which is a method to find maximum a posteriori estimates of parameters in a statistical model.
To achieve this, the algorithm makes use of the Forward-Backward algorithm.



So we have two steps, the E-step, 'E' stands for expectation, re-estimating the responsibility profile π given the current HMM parameters.
(emitted string , ? , Parameters) $\rightarrow \pi$



The other step is the M-step, 'M' stands for maximization, re-estimating the HMM parameters given the current responsibility profile π.
(emitted string, π, ?) $\rightarrow$ Parameters

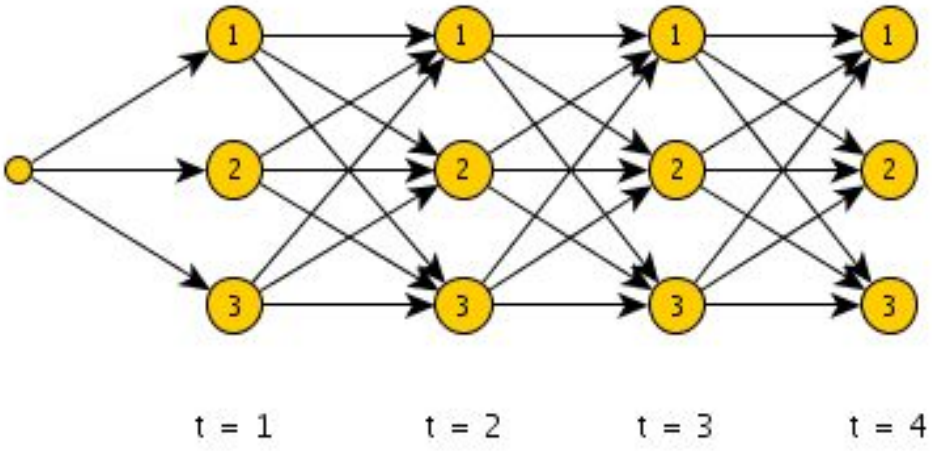
The Baum-Welch algorithm is needed because the state paths are hidden, and the equations cannot be solved analytically.

Conclusion:
The Baum-Welch algorithm attempts to find the model that assigns the training data the highest likelihood.

Viterbi algorithm

The Viterbi algorithm is most useful when one wants to calculate the most likely path through the state transitions of these models over time. Let's say we have N states and T moments in time, calculating the probabilities of all transitions over time would be N^T probability calculations. This algorithm is used to speed all these calculations up.

The observation made by the Viterbi algorithm is that for any state at time T, there is only one most likely path to that state. Therefore, if several paths converge at a particular state at time T, instead of redoing them all when calculating the transitions from this state to states at time T+1, one can discard the less likely paths, and only use the most likely one.



Conclusion:

Using the Viterbi algorithm we can identify the most likely sequence of hidden states given the sequence of observations.

Part II: Programming

Stock price prediction using Python

In this section, we'll talk about the program of which the main goal was to predict stock prices using a Hidden Markov Model to fit the data, and we'll give some detailed explanations on the functioning of the code.

```
import datetime
import numpy as np
from pandas_datareader import data
from hmmlearn.hmm import GaussianHMM
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from matplotlib.dates import YearLocator, MonthLocator
import warnings
warnings.filterwarnings('ignore')

ticker = 'EA'
start_date = datetime.date(2019, 1, 1)
end_date = datetime.date.today()
Stocks = data.DataReader(ticker, 'yahoo', start_date, end_date)

Stocks.reset_index(inplace=True,drop=False)
Stocks.drop(['Open','High','Low','Adj Close','Volume'],axis=1,inplace=True)
Stocks['Date'] = Stocks['Date'].apply(datetime.datetime.toordinal)
Stocks = list(Stocks.itertuples(index=False, name=None))

dates = np.array([q[0] for q in Stocks], dtype=int)
close_val = np.array([q[1] for q in Stocks])
close = [close_val[j]/close_val[j-1] for j in range(1, len(close_val))]

fig, ax = plt.subplots(figsize=(12,8))
plt.title('Stock Price over Time (EA)', fontsize = 14)
ax.plot_date(dates, close_val, '-', label='Close')
plt.gca().xaxis.set_major_locator(MonthLocator())
plt.xlabel('Date', fontsize = 14)
plt.ylabel('Close (price)', fontsize = 14)
plt.legend()
ax.autoscale_view()
ax.grid(True)
fig.autofmt_xdate()
plt.show()
```

First of all we have to import several packages in order to be able to run the code properly. The most important ones are the ‘‘pandas_datareader’ package, which is used to get our stock data from the ‘Yahoo Finance’ website, as well as the ‘hmmlearn.hmm’ package, which is used to fit the data with the Hidden Markov Model.

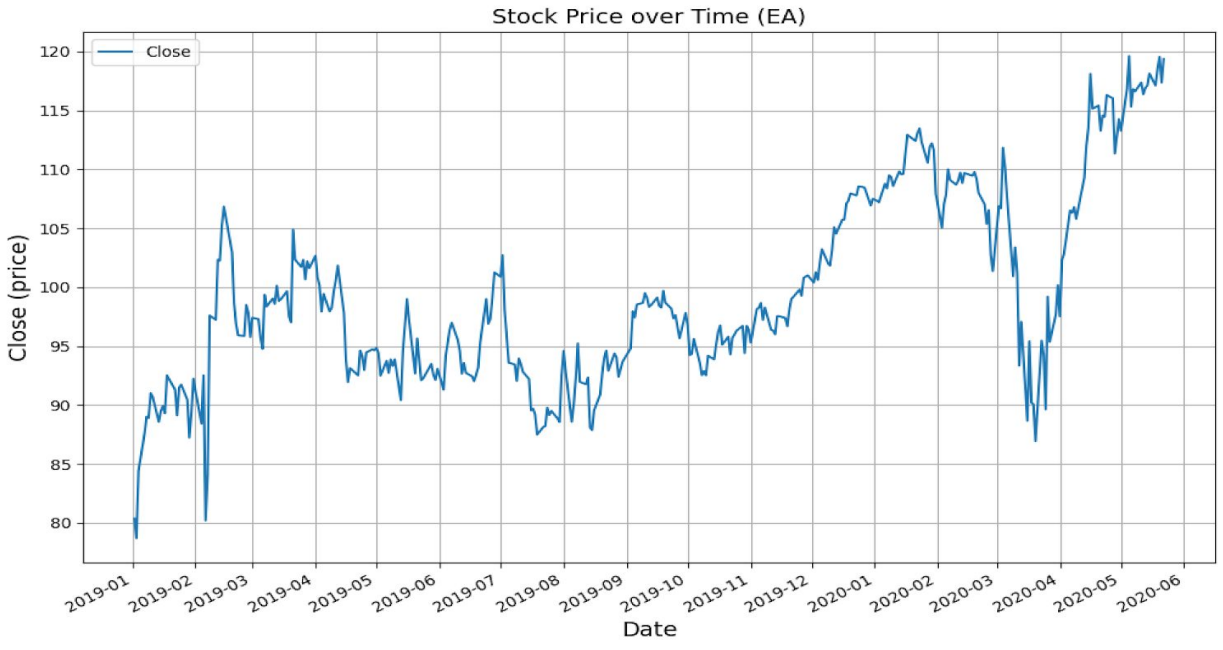
As mentioned above, the stock data that we use comes from the company ***Electronic Arts.*** We collect the data from the 1st January 2019 until the current date and convert it into a table using the ‘data.datareader()’ function.

This will look like that:

```
========== RESTART: C:\Users\belmi\Desktop\final.py ==========
                 High         Low        Open       Close    Volume   Adj Close
Date
2019-01-02   80.639999   77.199997   77.220001   80.370003  3176600   80.370003
2019-01-03   80.349998   77.620003   79.449997   78.709999  3747100   78.709999
2019-01-04   85.250000   80.040001   80.820000   84.419998  7254400   84.419998
2019-01-07   88.099998   83.440002   84.889999   87.589996  6611300   87.589996
2019-01-08   89.739998   87.690002   88.459999   89.019997  6468300   89.019997
...                ...         ...         ...         ...      ...         ...
2020-05-18  117.919998  114.730003  116.709999  117.120003  4336300  117.120003
2020-05-19  119.370003  116.720001  116.769997  118.540001  2517900  118.540001
2020-05-20  121.760002  119.059998  119.099998  119.550003  3530400  119.550003
2020-05-21  119.879997  117.199997  119.879997  117.360001  2257100  117.360001
2020-05-22  119.190002  116.614998  117.220001  119.070000   731565  119.070000

[351 rows x 6 columns]
    Returns
```

Now we rearrange the table. We only want to have the data that we'll need later on. That means we remove every column except the one with the closing prices of each day, which is the 'Close' column and the 'Date' column containing the dates.
Then we transform the dates into Gregorian ordinals which are representing the number of days passed since 1/1/0001. So for example the first day of our data, the 2nd of January 2019 (on the 1st of January the stock market is closed) would be represented by the number 737061.
After that we transform the table into a list containing the Gregorian ordinals and the closing prices for each day. We also define our lists 'dates' which are the Georgian ordinals, 'close_val' which are the closing prices and 'close'. For the 'close' list we take the closing price of a given day and divide it by the closing price of the previous day in order to have an indicator of the current stock situation.
At last, we plot the closing prices and get the following plot:



We will now proceed to the actual ‘prediction’ part of the program.

```
N = 5
train_x =  list(map(lambda el:[el], close[:N]))

model = GaussianHMM(n_components=2, covariance_type='diag', n_iter=1000)

predicted_prices = []
predicted_dates = []

for idx in range(100):
    model.fit(train_x)
    state = model.predict(train_x)[-1]
    means = model.means_[state]
    current_price = Stocks[N+idx][1]
    current_date = datetime.date.fromordinal(dates[N+idx])
    predicted_date = current_date + datetime.timedelta(days=1)
    predicted_dates.append(predicted_date)
    train_x.append([means[0]])
    predicted_prices.append(current_price * means[0])
    train_x.pop(0)


print('Actual:',close_val[N+1:-246], '\n','Prediction:', predicted_prices )

plt.figure(figsize=(12, 8))
plt.title('Close (price)', fontsize = 14)
plt.plot(predicted_dates,close_val[N+1:-246])
plt.plot(predicted_dates,predicted_prices)
plt.legend(['Actual','Predicted'])
plt.grid(True)
plt.show()
```

We start by defining the set of our data which will be used to train our model. For reasons of simplification we only use the first 5 days of our data as the training set. Then we convert every element of our training set ‘train_x’ into a list, so that we get a list of lists. This step is necessary in order to be able to train our model on our data.
The model we use is the ‘GaussianHMM’ model which uses the algorithms explained earlier in this report (Forward-Backward, Baum-Welch and Viterbi). As already mentioned, we train the model on the set ‘train_x’.

Now we define the 2 lists ‘predicted_prices’ and ‘predicted_dates’ that are evidently empty in the beginning. In order to fill these lists, we’re using a for loop with range 100.
We start the loop by fitting our model and then looking at the hidden states. As we only want to predict the price for 1 day in the future, we just look at what the hidden state corresponds to for the last day of the training set. As an example, let’s say that for the last day the hidden state corresponds to 1. Then we look at the mean parameter of the hidden state corresponding to the one we found (our expected return), so in that case we would be getting the mean parameter for the hidden state 1.

As we want to compare the actual prices with our predicted prices, we first set our current date to be the last day of our training set and look at the closing price for that day. Our predicted date will then be the following day.
Now we want to update our training set after every run of the loop. We add to the list the element '[means[0]]' and we remove the first element.
Then we just need to calculate our predicted price by multiplying the current price with our expected returns.
This process gets repeated 100 times in order to predict the prices for 100 days.

Finally we print and plot our actual and our predicted prices.