Markov Model - An Introduction

In this post, we will learn about Markov Model and review two of the best known **Markov Models** namely the **Markov Chains**, which serves as a basis for understanding the Markov Models and the **Hidden Markov Model (HMM)** that has been widely studied for multiple purposes in the field of forecasting and particularly in trading.

In this post we will try to answer the following questions:

We will also see how to implement some of these ideas with Python that will serve as a basis for experimentation.

What is a Markov Model?

A Markov Model is a set of mathematical procedures developed by Russian mathematician Andrei Andreyevich Markov (1856-1922) who originally analyzed the alternation of vowels and consonants due to his passion for poetry.

In the paper that E. Seneta [1] wrote to celebrate the 100th anniversary of the publication of Markov's work in 1906 [2], [3] you can learn more about Markov's life and his many academic works on probability, as well as the mathematical development of the Markov Chain, which is the simplest model and the basis for the other Markov Models.

In the late 1960s and early 1970s Leonard E. Baum and his colleagues studied, developed and extended the Markov techniques by creating new models such as the Hidden Markov Model (HMM) [4].

What are Markov Models used for?

Nowadays Markov Models are used in several fields of science to try to explain random processes that depend on their current state, that is, they characterize processes that are not completely random and independent. They are also not governed by a system of equations where a specific input corresponds to an exact output.

A deterministic model attempts to explain with precision and accuracy the behaviour of a process and a probabilistic or stochastic model attempts to determine by probability the behavior of a randomized independent process. In contrast, the Markov Model attempts to explain a random process that depends on the current event but

not on previous events, so it is a special case of a probabilistic or stochastic model.

When we have a dynamic system whose states are fully observable we use the Markov Chain Model and if the system has states that are only partially observable we use the Hidden Markov Model.

For example, a dynamic system can be a price stream with a certain frequency, either minutes, hours, days or weeks or with an undetermined frequency such as ticks, which has observable states, such as if the price goes up, down or unchanged, although it can also be a price stream or a certain price figure. We will go into detail when we see how the Markov Chain works.

The Markov Model uses a system of vectors and matrices whose output gives us the expected probability given the current state, or in other words, it describes the relationship of the possible alternative outputs to the current state.

How does a Markov Model work?

Let's start by naively describing how the simplest model, Markov Chain works. In this post, we are going to focus on some implementation ideas in Python but we are not going to stop at the formulation and mathematical development.

It is recommended to the interested reader to review the tutorial on Hidden Markov Models and Selected Applications in Speech Recognition by Lawrence R. Rabiner [6] to get a solid base on the mathematical foundations of the Markov Chain and the HMM.

With a Markov Chain, we intend to model a dynamic system of observable and finite states that evolve, in its simplest form, in discrete-time. We can describe it as the transitions of a set of finite states over time.

Let's take a simple example to build a Markov Chain. Let's say we have a series of SPY prices and we want to model the behavior to make predictions about the future price.

To do this, we need the frequency distribution of each possible state in time t. From this, we generate a transition matrix or probability matrix that can be multiplied iteratively by the original transition matrix, which allows us to extend the behaviour of the model in time and to know the probability of the states in time t+i.

The Markov Chain reaches its limit when the transition matrix achieves the equilibrium matrix, that is when the

multiplication of the matrix in time t+k by the original transition matrix does not change the probability of the possible states.

Let's get the 2018 prices for the SPY ETF that replicates the S&P 500 index.

```
import numpy as np
import pandas as pd
#import tensorflow as tf
import yfinance as yf
import talib as ta
```

```
#SPY = yf.Ticker("SPY", start='2010-01-01').history(period="max")
SPY = yf.download("SPY", start="2018-01-01", end="2018-12-31")
```

```
[**********************100%**********************]  1 of 1 completed
```

```
data = SPY[['Open', 'High', 'Low', 'Adj Close']].copy()
data.tail()
```

Date	Open	High	Low	Adj Close
2018-12-21	246.740005	249.710007	239.979996	236.226837
2018-12-24	239.039993	240.839996	234.270004	229.985046
2018-12-26	235.970001	246.179993	233.759995	241.605011
2018-12-27	242.570007	248.289993	238.960007	243.459900
2018-12-28	249.580002	251.399994	246.449997	243.145813

The first thing is to identify the states we want to model and analyze. In this example, we will simply consider whether the price moves up, down or is unchanged.

To summarize, our three possible states are:

1. Up: The price has increased today from yesterday's price.
2. Down: the price is decreased today compared to yesterday's price
3. Flat: The price remains unchanged from the previous day.

To obtain the states in our data frame, the first task is to calculate the daily return, although it should be remembered that the logarithmic return is usually better fitted to a normal distribution.

```
data['pct_ret'] = data['Adj Close'].pct_change()
data.tail()
```

Date	Open	High	Low	Adj Close	pct_ret
2018-12-21	246.740005	249.710007	239.979996	236.226837	-0.020490
2018-12-24	239.039993	240.839996	234.270004	229.985046	-0.026423
2018-12-26	235.970001	246.179993	233.759995	241.605011	0.050525
2018-12-27	242.570007	248.289993	238.960007	243.459900	0.007677
2018-12-28	249.580002	251.399994	246.449997	243.145813	-0.001290

We then identify the possible states according to the return. The Flat state could be defined as a range and hence to consider an up/down as a minimum movement.

```
# compute the states
data['state'] = data['pct_ret'].apply(lambda x: 'Up' if (x > 0.001)
                                      else ('Down' if (x < -0.001)
                                            else 'Flat' ))
```

```
data.tail()
```

Date	Open	High	Low	Adj Close	pct_ret	state
2018-12-21	246.740005	249.710007	239.979996	236.226837	-0.020490	Down
2018-12-24	239.039993	240.839996	234.270004	229.985046	-0.026423	Down
2018-12-26	235.970001	246.179993	233.759995	241.605011	0.050525	Up
2018-12-27	242.570007	248.289993	238.960007	243.459900	0.007677	Up
2018-12-28	249.580002	251.399994	246.449997	243.145813	-0.001290	Down

We are interested in analyzing the transitions in the prior day's price to today's price, so we need to add a new column with the prior state.

```
data['priorstate'] = data['state'].shift(1)
data.tail()
```

Date	Open	High	Low	Adj Close	pct_ret	state	priorstate
2018-12-21	246.740005	249.710007	239.979996	236.226837	-0.020490	Down	Down
2018-12-24	239.039993	240.839996	234.270004	229.985046	-0.026423	Down	Down
2018-12-26	235.970001	246.179993	233.759995	241.605011	0.050525	Up	Down
2018-12-27	242.570007	248.289993	238.960007	243.459900	0.007677	Up	Up
2018-12-28	249.580002	251.399994	246.449997	243.145813	-0.001290	Down	Up

With the current state and the prior state, we can build the frequency distribution matrix.

```
# Frequency distributions
states = data[['priorstate', 'state']].dropna()
states_mat = states.groupby(['priorstate', 'state']).size().unstack()
states_mat
```

priorstate \ state	Down	Flat	Up
Down	46	6	50
Flat	18	3	11
Up	39	22	54

Here we have gotten the frequency distribution of the transitions, which allows us to build the initial probability matrix or transition matrix at time t0.

```
# Initial transition matrix
transition_matrix = states_mat.apply(lambda x: x / float(x.sum()), axis=1)
transition_matrix
```

priorstate \ state	Down	Flat	Up
Down	0.45098	0.058824	0.490196
Flat	0.56250	0.093750	0.343750
Up	0.33913	0.191304	0.469565

This would be our transition matrix in t0, we can build the Markov Chain by multiplying this transition matrix by itself to obtain the probability matrix in t1 which would allow us to make one-day forecasts.

```
# find the transition matrix at t1
t0 = transition_matrix.copy()
t1 = round(t0.dot(t0),4)
t1
```

state	Down	Flat	Up
priorstate			
Down	0.4027	0.1258	0.4715
Flat	0.4230	0.1076	0.4694
Up	0.4198	0.1277	0.4525

If we continue multiplying the transition matrix that we have obtained in t1 by the original transition matrix in t0, we obtain the probabilities in time t2.

```
# find the transition matrix at t2
t2 = round(t0.dot(t1),4)
t2
```

state	Down	Flat	Up
priorstate			
Down	0.4123	0.1257	0.4621
Flat	0.4105	0.1247	0.4648
Up	0.4146	0.1232	0.4622

Multiplying the transition matrix that we have obtained in t2 by the original transition matrix in t0, we obtain the probabilities in time t3 and so on until we find the equilibrium matrix where the probabilities do not change and therefore we cannot continue evolving the prediction.

```
# find the transition matrix at t3
#t3 = round(t0.dot(t2),10)
t3 = t0.dot(t2)
t3
```

state	Down	Flat	Up
priorstate			
Down	0.413316	0.124425	0.462259
Flat	0.412912	0.124752	0.462336
Up	0.413031	0.124355	0.462615

Interestingly, you can get out identical results by raising the initial transition matrix to ‘n’ days to obtain the same result.

```
pd.DataFrame(np.linalg.matrix_power(t0, 4))
```

	0	1	2
0	0.413316	0.124425	0.462259
1	0.412912	0.124752	0.462336
2	0.413031	0.124355	0.462615

To find out the equilibrium matrix we can iterate the process up to the probabilities don’t change more.

```
# find the equilibrium matrix
i = 1
a = t0.copy()
b = t0.dot(t0)
while(not(a.equals(b))):
    print("Iteration number: " + str(i))
    i += 1
    a = b.copy()
    b = b.dot(t0)
```

```
Iteration number: 1
Iteration number: 2
Iteration number: 3
Iteration number: 4
Iteration number: 5
Iteration number: 6
Iteration number: 7
Iteration number: 8
Iteration number: 9
Iteration number: 10
Iteration number: 11
Iteration number: 12
Iteration number: 13
Iteration number: 14
Iteration number: 15
Iteration number: 16
Iteration number: 17
Iteration number: 18
Iteration number: 19
Iteration number: 20
Iteration number: 21
Iteration number: 22
```

With this example, we have seen in a simplified way how a Markov Chain works although it is worth analyzing the different libraries that exist in Python to implement the Markov Chains.

What is the Hidden Markov Model?

The Hidden Markov Model (HMM) was introduced by Baum and Petrie [4] in 1966 and can be described as a Markov Chain that embeds another underlying hidden chain.

The mathematical development of an HMM can be studied in Rabiner's paper [6] and in the papers [5] and [7] it is studied how to use an HMM to make forecasts in the stock market.

In this blog, we explain in depth, the concept of Hidden Markov Chains and demonstrate how you can construct Hidden Markov Models.

Also, check out this article which talks about Monte Carlo methods, Markov Chain Monte Carlo (MCMC).

If you want to detect a Market Regime with the help of a hidden Markov Model then check out this EPAT Project.

What is the difference between the Markov Model and the Hidden Markov Model?

As we have seen a Markov Model is a collection of mathematical tools to build probabilistic models whose current state depends on the previous state.

This is the initial view of the Markov Chain that later extended to another set of models such as the HMM.

The HMM is an evolution of the Markov Chain to consider states that are not directly observable but affect the behaviour of the model.

So, we learnt about Markov Chains and the Hidden Markov Model (HMM). I'd really appreciate any comments you might have on this article in the comment section below. Do feel free to share the link of this article. I've also provided the Python code as a downloadable file below.

References:

- [1] Seneta, Eugene. (2006). Markov and the creation of Markov Chains.
- [2] A.A. Markov, Extension of the law of large numbers to dependent quantities (in Russian), Izvestiia Fiz.-Matem. Obsch. Kazan Univ., (2nd Ser.), 15(1906), pp. 135–156 pp.
- [3] A.A. Markov, The extension of the law of large numbers onto quantities depending on each other. Translation into English of [2].
- [4] Baum, Leonard E., and Ted Petrie. 1966. Statistical inference for probabilistic functions of finite state Markov chains. The Annals of Mathematical Statistics 37: 1554–63.
- [5] Nguyen, Nguyet. 2018. Hidden Markov Model for Stock Trading.
- [6] Rabiner, Lawrence R. 1989. Hidden Markov Models and Selected Applications in Speech Recognition.
- [7] Hassan, Rafiul and Nath, Baikunth. 2005. Stock Market Forecasting Using Hidden Markov Model: A New Approach.

File in the download: Markov Model working - Python Code

Login to Download