

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



# Factorial Hidden Markov Model for Time Series Analysis in Python

5 min read · Feb 29, 2024



Erdal GENÇ

Follow



Listen



Share



More

For a start, I would like to give some intro about Factorial Hidden Markov Model (FHMM).

The Factorial Hidden Markov Model (FHMM) is an extension of the Hidden Markov Model (HMM) that allows for modeling of multiple time series with their interactions.

Traditional HMMs model a single time series with a hidden state variable that governs the generation of the observed variable at each time step. In contrast, an FHMM models multiple time series with multiple corresponding hidden state variables that interact to generate the observed variables.

FHMM assumes that each of the observed time series has a unique hidden state, but the states are conditionally independent given a common set of underlying states. Each hidden state factor in an FHMM generates a separate time series that is conditionally independent of the others, given the shared underlying states.

The shared underlying states in an FHMM converge the information from the individual factors to model the dependence and interaction between the time series. This convergence is achieved by computing the probabilities of each time series given each underlying state and then weighting those probabilities to compute the joint probability distribution over the set of observed variables.

FHMMs are commonly used in modeling multivariate time series data in various applications such as Financial Time Series Prediction and Portfolio Optimization.

First, we need to install `hmmlearn` package from PyPI: `pip install hmmlearn numpy`:  
<https://pypi.org/project/hmmlearn/>

Secondly, the necessary packages are imported: `hmmlearn` for the HMM, and `numpy` for array manipulation. The number of states and observations are set to 2 and 3, respectively.

```
from hmmlearn import hmm
import numpy as np
num_states = 2
num_obs = 3
```

Next, the transition matrix (i.e. the probability of transitioning from one state to another) and emission probabilities (i.e. the probability of emitting an observation given a state) are defined. `emit_means` is a list of mean emission probabilities for each state, and `emit_covars` is a list of covariance matrices for each state.

```
# Transition matrix
trans_mat = np.array([[0.5, 0.5], [0.5, 0.5]])

# Mean emission probabilities
emit_means = [np.array([1, 0, 0]), np.array([0, 1, 0]), np.array([0, 0, 1]),
               np.array([0.5, 0.5, 0]), np.array([0.5, 0, 0.5]), np.array([0, 0.5, 0.5])]

# Emission covariances
emit_covars = [np.eye(num_obs) for _ in range(num_states * 2)]
```

Open in app ↗

Medium



Search



`covariance_type` set to `'diag'`, meaning that the covariance matrices are diagonal.

The model is trained using the `fit()` method on `emit_means`.

```
fhmm = hmm.GaussianHMM(n_components=num_states * 2, covariance_type='diag', n_i  
  
fhmm.startprob_ = np.repeat(0.5, num_states * 2)  
fhmm.transmat_ = trans_mat  
  
fhmm.fit(emit_means)
```

More training sequences are generated with random observations using the `np.random` function and concatenated to create `X_concat`. The individual lengths of each sequence are calculated and used in the second `fit()` method along with `X_concat`.

```
# Generate more sequences for training  
X = [np.random.rand(np.random.randint(10, 20), num_obs) for _ in range(10)]  
  
# Concatenate the sequences along the first axis (rows)  
X_concat = np.concatenate(X)  
  
# Compute the individual sequence lengths and train the model  
lengths = [len(seq) for seq in X]  
fhmm.fit(X_concat, lengths=lengths)
```

Test data is generated in the same manner as the training sequences, and the `predict()` method is used to predict state sequences based on the model. New observations are generated using the `sample()` method with `len(X_concat_test)` as the number of samples to generate.

```
X_test = [np.random.rand(np.random.randint(10, 20), num_obs) for _ in range(10)]  
X_concat_test = np.concatenate(X)
```

Finally, the predicted state sequence and new observations are printed.

```
# Predict state sequence for new test data
state_sequence = fhmm.predict(X_concat_test)

# Generate observations based on learned model
new_obs = fhmm.sample(len(X_concat_test), random_state=0)

print("State sequence:", state_sequence)
print("New observations:", new_obs)
```

The use of Factorial Hidden Markov Models (FHMMs) in time series analysis has several **advantages** over other models:

1. **Model flexibility:** FHMMs can model time series with complex patterns and dependencies, as they are capable of modeling individual time series as well as the relationships between them. This makes them suitable for a wide range of time series data, including multivariate, heterogeneous and multidimensional time series.
2. **Efficient estimation:** FHMMs can be estimated efficiently with an Expectation-Maximization algorithm that can be implemented in parallel, making it suitable for large and high-dimensional time series.
3. **Interpretability:** FHMMs allow for easy interpretation and visualization of the results. The model decomposes the time series into a set of underlying components, each of which can be separately modeled and analyzed. This helps to identify trends, patterns and correlations in the data.
4. **Generative modeling:** FHMMs allow for the generation of new time series data that follows the learned structure, which can be useful for predicting future trends and making forecasts.
5. **Robustness:** FHMMs are robust to missing data and can handle incomplete observations. This ability enables FHMMs to be used for tasks like imputation, where missing values can be replaced with expected values predicted by the model.

Overall, these advantages make FHMMs a powerful and flexible tool for time series analysis, which can improve the accuracy of predictions and uncover hidden patterns in time series data.

While FHMMs have several advantages over other models in time series analysis, they also have some **drawbacks**. Here are some of the main **disadvantages** of FHMMs:

1. **Computational complexity:** Estimating an FHMM can be computationally expensive, especially when dealing with high-dimensional and multivariate time series data. This is because the model involves multiple layers of latent variables, and estimating the model parameters can be time-consuming.
2. **Model selection difficulty:** There is no universally optimal way to select the number of hidden states or the number of factors in an FHMM. The choice can vary depending on the data set, which can lead to difficulties in model selection. Sometimes poor choices of the number of hidden states can lead to overfitting or underfitting the model, which can reduce its accuracy.
3. **Model interpretability:** Although FHMMs offer interpretability in terms of the learned decomposition of the time series, the meaning of the individual components may not always be clear or intuitive.
4. **Sensitivity to initialization:** Estimation of FHMM parameters is sensitive to initialization, and different starting points may lead to different results. This can make it difficult to reproduce the model fitting process or compare models with different results.
5. **Assumption of underlying Markov model/gaussian assumption:** FHMMs make some assumptions about the underlying Markov process and Gaussian emission distributions, which may not always hold in practice. Violating these assumptions can lead to poor model performance.

In summary, while FHMMs offer a range of benefits, users also need to carefully consider their limitations in application and choose the most appropriate model for their specific use case.

Thank you for reading!

Markov Models

Python

Time Series Analysis



Follow

## Written by Erdal GENÇ

20 Followers · 5 Following

No responses yet



Wangchuyin

What are your thoughts?

## More from Erdal GENÇ





Erdal GENÇ

## Enhancing Financial Data Analysis with Python: 9 Feature Ranking Methods for Improved Model...

Feature importance for financial data analysis in Python

Aug 14, 2024

👏 25

💬 2



**https://quay.io**

The URL to which the application will link in the authorization view

**Description (optional)**

**Generate Token**

Description

The user friendly description of the application

**Avatar E-mail (optional)**

Avatar E-mail

An e-mail address representing the avatar for the application. See above for the icon.



Erdal GENÇ

## Github Action to Automate build & push for Dockerfiles Repository to Quay.io Docker Registry

This article is for automation of docker build and push to the Quay.io registry in a Docker-containers repo.

Aug 15, 2022

👏 5



# er - Quick Options

[Go to advanced options](#)

## General Configuration

Cluster name

☒ Login

S3 fold

Launch mode ☒ Cluster

 Erdal GENÇ

### Spark-NLP for Healthcare in AWS EMR

In this article, we explain how to setup Spark-NLP + Spark-NLP Healthcare in AWS EMR, using the AWS console. This configuration is already...

Mar 3, 2022  52



 Erdal GENÇ

### Exploratory Data Analysis with Web App using Streamlit

Here is the full repo: <https://github.com/egenc/data-visualization-streamlit>

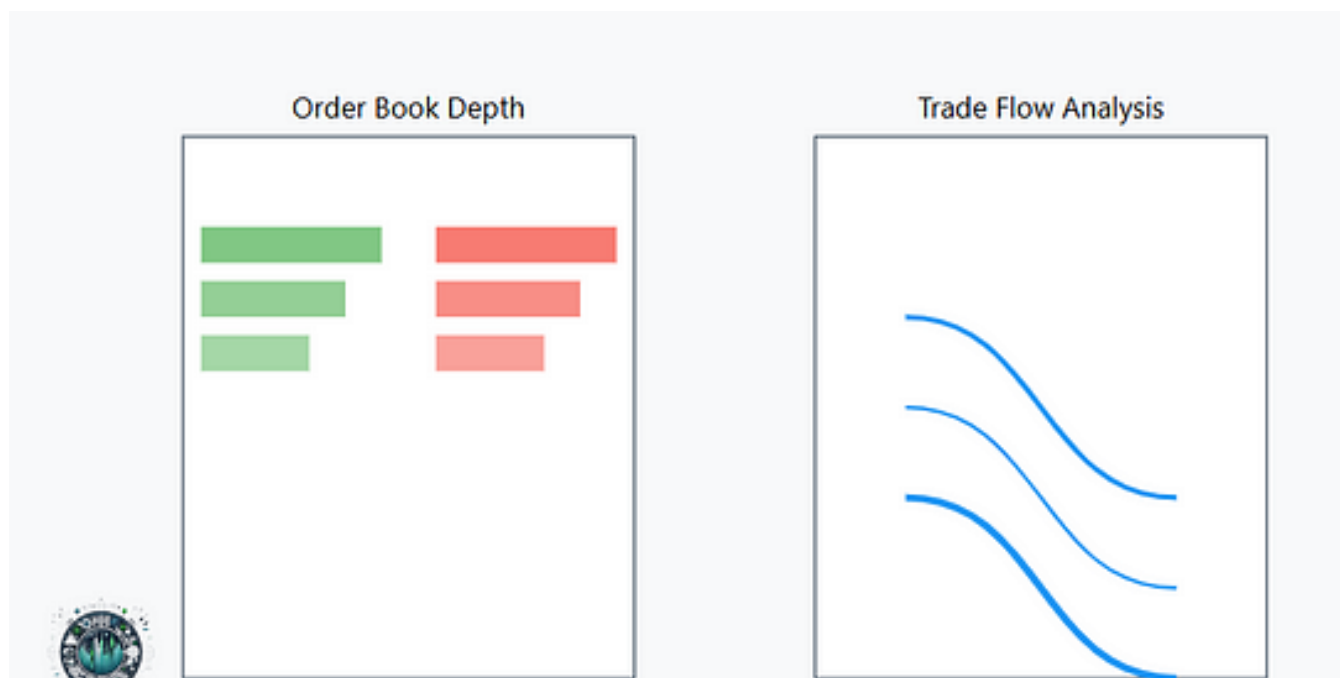



Apr 5, 2023



See all from Erdal GENÇ

## Recommended from Medium



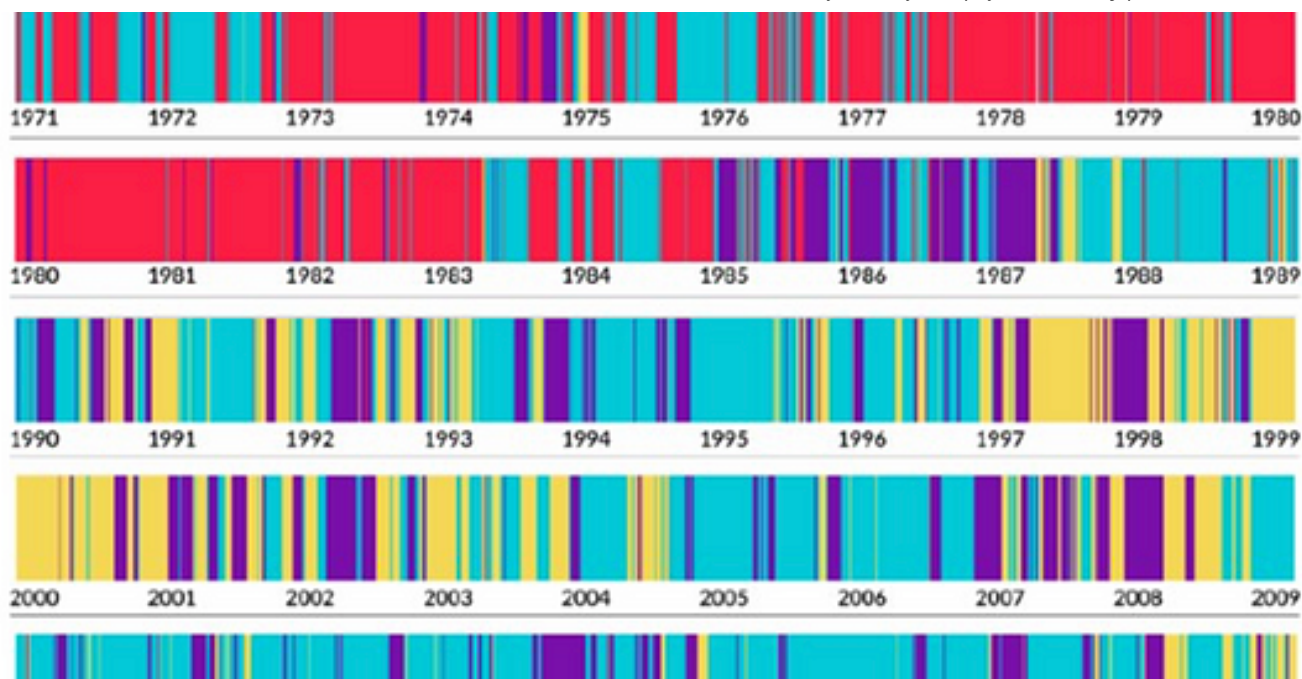
 In Funny AI & Quant by Pham The Anh

## Unlocking Alpha: Advanced Exploratory Data Analysis Techniques in Quantitative Trading

In the dynamic realm of quantitative trading, the ability to extract meaningful insights from financial data sets the foundation for...

★ Dec 27, 2024  1





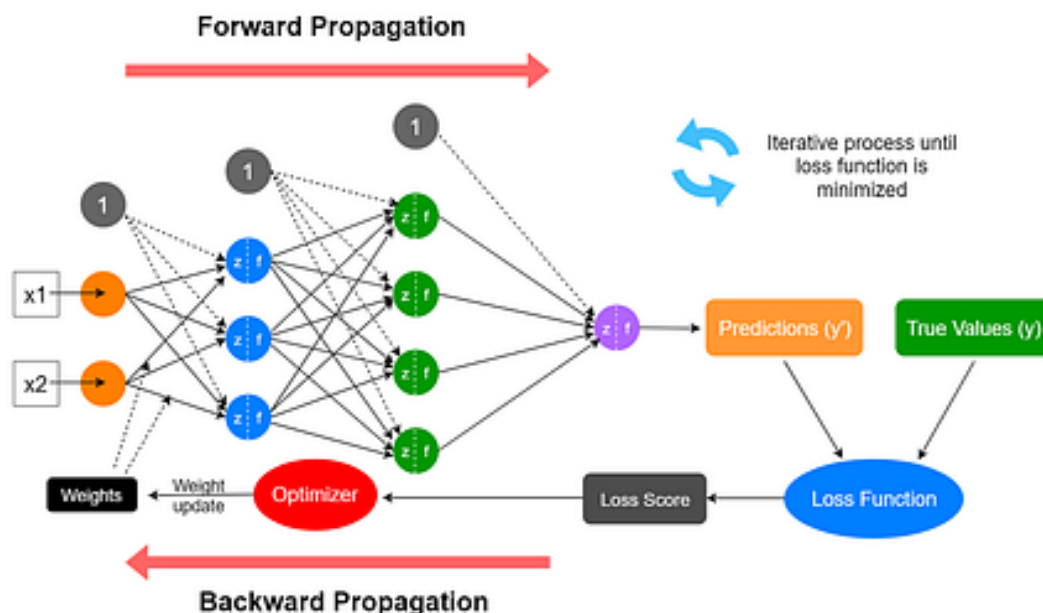
 Big Universe Little Time

## A Hidden Markov Model Analysis for Gold Historical Data

What is “Hidden States” concept? Why are the states “Hidden” and not observable? How can we foresee the states in Candlestick Chart data...

🌟 Nov 4, 2024 🖱️ 35

🔖 ...



 In Towards Explainable AI by Sandipan Paul

## Neural Network In SHORT

A neural network is composed of layers of interconnected neurons that process information. The primary components are the Input layer which...

★ Apr 8 🖱 69 💬 1

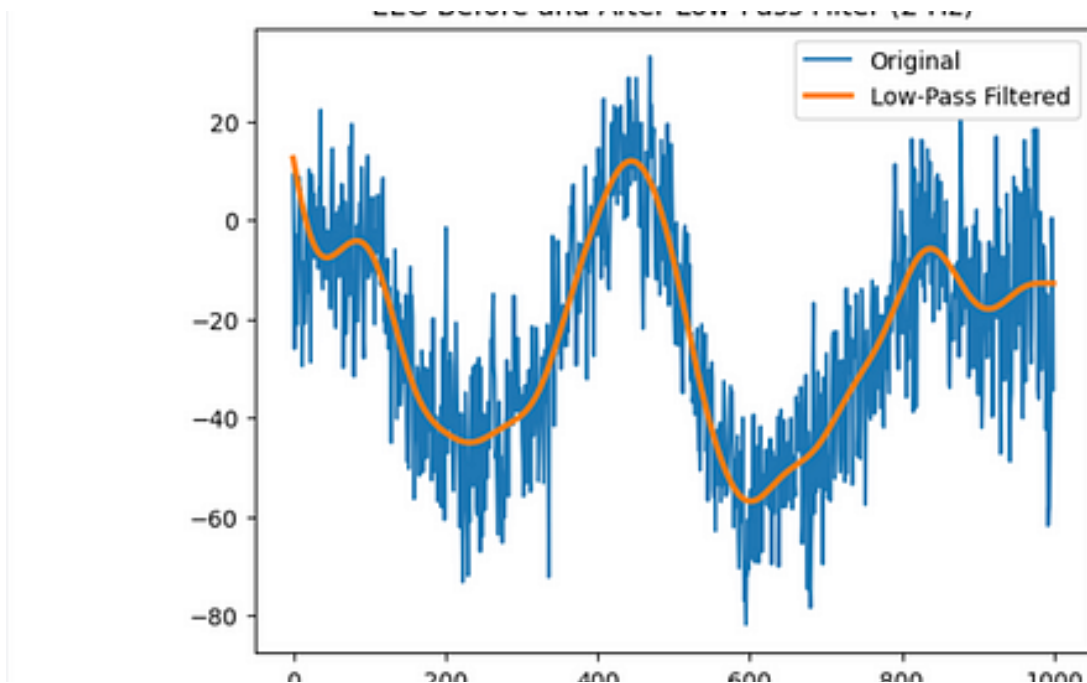


In InsiderFinance Wire by Tony Tsoi

## Option Pricing with Monte Carlo Method –The Martingales Theory

Pricing of Path-Dependent Exotic Options

Dec 10, 2024 🖱 66



Proto Bioengineering

## Digital Signals for Dumb\*sses (Part 6: How to Remove Frequencies from a Signal with Python)

Remove noise, smooth signals, or pluck out certain frequencies with Python and SciPy

★ Dec 28, 2024



 Aakash Chavan Ravindranath, Ph.D

## Hidden Markov Model and the Medilian Fund: A Deep Dive into Market Prediction

Introduction In finance, making informed predictions is key to successful trading. One powerful tool that helps in this endeavour is the...

★ Mar 2 🤝 1



See more recommendations