Time Series Regime Analysis in Python

Spencer · Follow
13 min read · Oct 13, 2022

This is an introductory article to time series regime analysis in python. My aim is to demonstrate how to detect and predict regimes in time series, with the application tailored to financial time series. Find the full notebook and code for this article here.

By definition, the word regime suggests a system or structure determining how things are done. A regime shift in time series refers to the changing behavior of the time series in different time intervals. In financial time series, price data is merely a representation of the activities and views of participants. Price changes as orders are placed and filled in an order book, these orders represent the market opinion and subsequent bet of each participant. A regime shift in financial time series can be thought of as the changing behavioral structure of market participants.

To highlight the importance of time series regime analysis, one does not have to look further than classic time series forecasting techniques. A linear autoregressive (AR) model for example, suggests that a value in time has a linear dependency on its previous values and on an error term. An autoregressive model of order p is represented below.

$$x_t = c + \sum_{i=1}^{p} a_i x_{t-i} + \epsilon_t$$

[1]

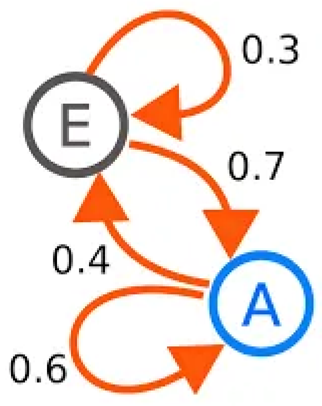
The main assumption of the AR approach is that the mean and variance of the time series remains constant. We may consider this to be a single regime in terms of mean and variance. In reality time series do not always present with constant regimes and series are often observed changing regimes. This necessitates the understanding and detection of underlying regimes to improve forecasting techniques.

There are a variety of approaches to understanding time series regimes. The approach presented in this article is the use of a Markov Switching Autoregressive model for regime detection and classification. Furthermore, machine learning is explored for providing an expectation of the next day's regime.

. . .

Markov Switching Autoregression

A Markov Switching model is a popular regime-switching model that rests on the assumption that unobserved states are determined by an underlying stochastic process known as a Markov-chain. A Markov-chain is a system describing possible events in which the probability of each event depends only on the state attained in the previous event. The image below provides a simple example of a Markov-chain with two states (regimes) E and A. The probability of remaining in state E is 0.3. The probability of transition to state A given that we are in state E is 0.7. Notice that the sum of these two transition probabilities, related to E, is 1.



[2]

Knowing the true transition probabilities, particularly in financial time series, would assume perfect knowledge of the environment. That is unfortunately not the case so we need a way of estimating these probabilities and characteristics. Transition probabilities and state characteristics are usually estimated using maximum likelihood estimation or Bayesian estimation.

Markov Switching Autoregressive Models (MS-AR) still define the price of an asset as an autoregressive process, however each process is also regime specific. Representing this mathematically, an easy to understand MS-AR process with 3 regimes (S) is shown below.

$$r_t = \begin{cases} a_1 + \beta_{11} r_{t-1} + \cdots + \beta_{p1} r_{t-p} + \varepsilon_t & S_t = 1 \\ a_2 + \beta_{12} r_{t-1} + \cdots + \beta_{p2} r_{t-p} + \varepsilon_t & S_t = 2 \\ a_3 + \beta_{13} r_{t-1} + \cdots + \beta_{p3} r_{t-p} + \varepsilon_t & S_t = 3 \end{cases}$$

[3]

• • •

Now moving onto the implementation, this will be separated into two sub-sections. The first section deals with the MS-AR implementation for regime detection and classification. The second section deals with the machine learning, Random Forest Classifier, implementation for the prediction of the next regime.

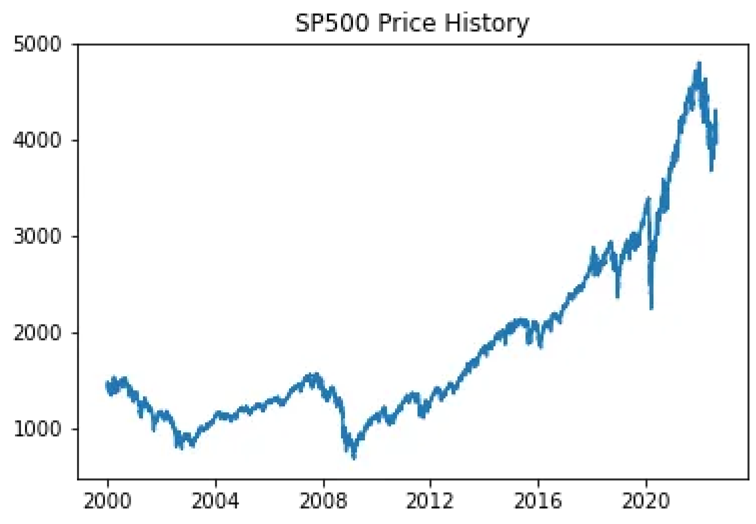
MS-AR

First of all, we are going to need data. The Standard & Poor's 500 (S&P500) index of daily frequency is used. The S&P500 is a stock market index tracking the performance of 500 large companies listed on stock exchanges in the United States. We use the S&P 500 ticker ^GSPC, specifying the start and end dates as well as a daily interval. The data is sourced from Yahoo finance.

```python
df = yf.download('^GSPC', start="2000-01-01", end="2022-09-01", interval="1d")
df.head()
```

download_data.py hosted with ❤ by GitHub view raw

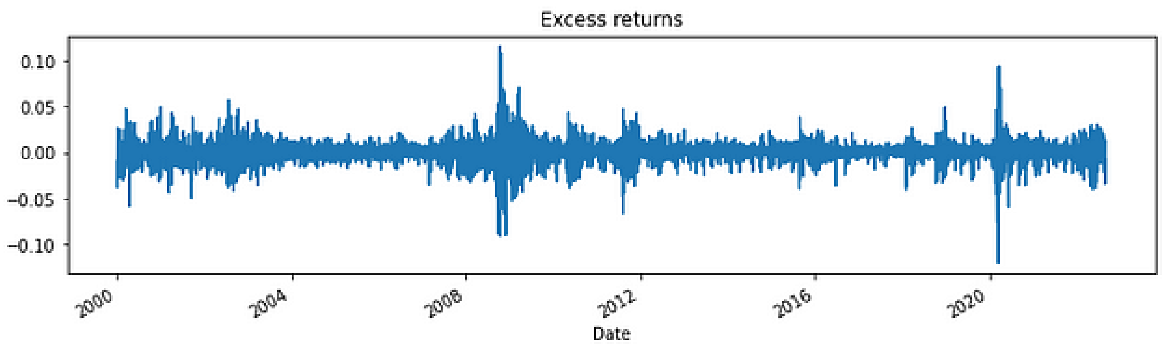
Plotting the adjusted close prices gives us a good picture of what we are dealing with. This is clearly an increasing series with drawdowns around major crises such as the global financial crisis of 2008/2009 and the COVID-19 pandemic of 2020.



Before fitting our model it's important to understand what we are trying to achieve with regime analysis. If we were investing in a fund tracking the S&P 500 index, what would we do? We may buy and hold, making a handsome return over the last 22 years. However, what would be even better would be to avoid significant drawdowns. If we had a way of characterizing the time series during these periods of losses, hopefully exiting the market and only re-entering after the drawdown period.

Now, above I mentioned that if we could characterize the time series, we may be able to identify and avoid large drawdowns. The word characterize leads into one of the most important parts of the article. A simple characteristic describing how price is changing is the percentage change in price from one period to the next. We will call this the price return. Referring back to our discussion on what a market is, price is changing because participants are buying and selling an asset. Therefore, the percentage return of an asset can be loosely thought of as characterizing the buying and selling behavior of market participants. In reality there is a lot more that could characterize this behavior, like traded volumes or high-low spreads. But for simplicity's sake, let's take one period price returns as our only characteristic. When markets are calm, participants are not frantically buying and selling assets so daily price returns vary less. Prices also tend to trend upward. We may describe price returns as having a low variance, suggesting that there is a relatively low degree of spread in returns. When markets are in a state of panic, participants are frantically buying and selling assets to either make short term gains or avoid losses. We may describe price returns as having a high variance as there is a high spread in daily returns. This panic would likely start occurring when assets are becoming overpriced or in response to a marco-event.

Given the discussion above, a plot of the price returns (excess returns) starts to paint a picture of possible underlying regimes and how returns could characterize participants behavior.



Focusing on our key dates, 2008/2009 and 2020, we can see price returns spike dramatically. This supports the hypothesized dynamics we alluded to previously. Furthermore, supporting the use of price returns to characterize the behavior of markets and participants, suggesting that there are in fact underlying regimes described by price return variance.

```python
mod_kns = sm.tsa.MarkovRegression(ex_ret.dropna(), k_regimes=2, trend='n', switching_variance=Tr
res_kns = mod_kns.fit()
res_kns.summary()
```

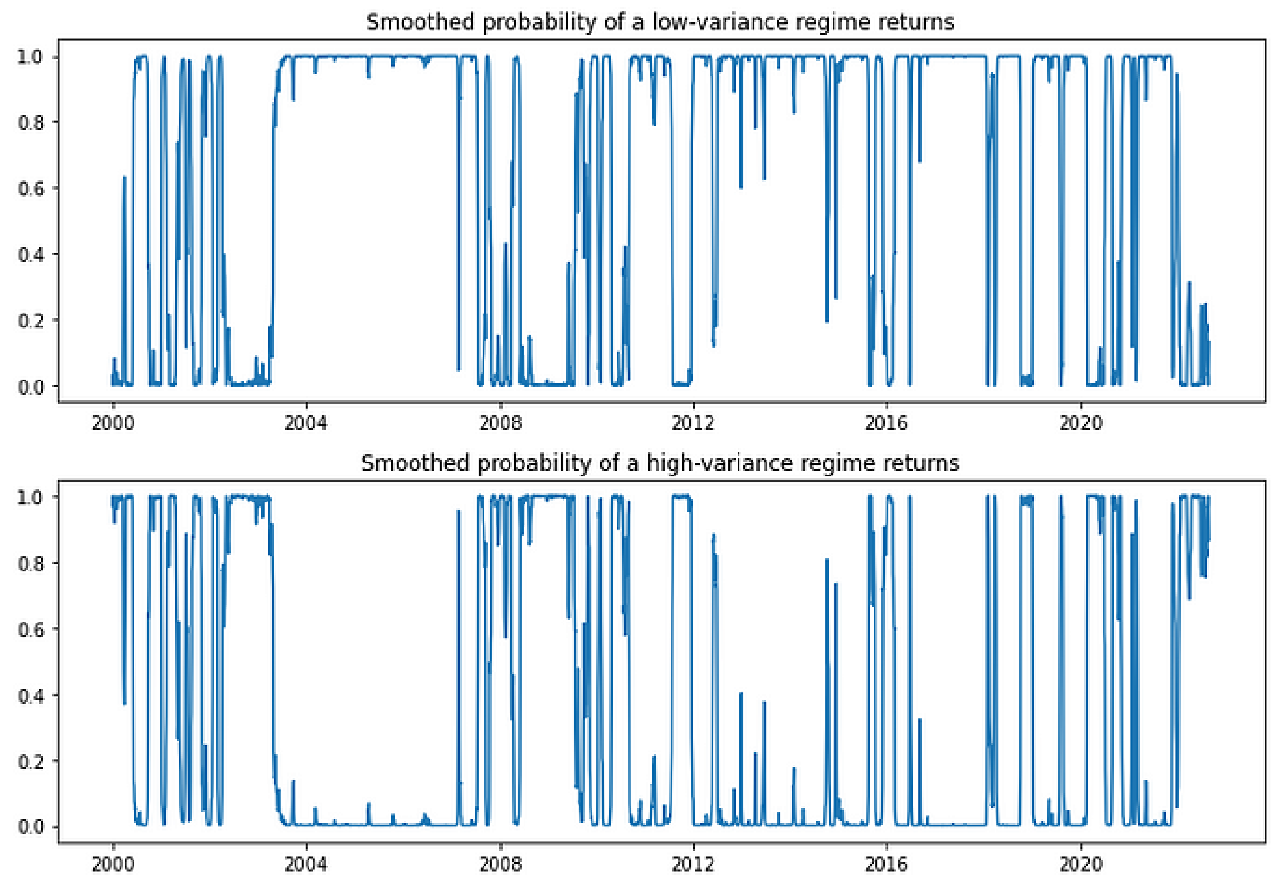
model_fitment.py hosted with ❤ by GitHub view raw

Fitting the markov regression is the easiest part of the analysis. We instantiate the model and fit it in only two lines of code. The final line of code will give us a summary of fitment results. When creating the model we specify the number of regimes we are wanting to detect. In my experience, trying to detect and characterize more than 3 or 4 regimes becomes tedious with the usefulness of results decaying rapidly. In this case we specify 2 regimes, these can be characterized as a low variance return regime and a high variance return regime. Trend is specified as no trend, as we are exploring stationary (or close to stationary) price return data. Variance switching being set to true means that we expect there to be regime specific heteroskedasticity. In other words the variance of our residuals is not constant, and we expect this to be regime specific.

Next we want to view our actual regime classifications in time. It is important to note that the raw output of a 2 regime Markov switching regression is two values bounded between 0 and 1. That suggests the likelihood that the current time period is a member of either regime. The output can be interpreted probabilistically. The first value is the model's probability expectation of low variance regime membership and the second value is the expectation of high variance regime membership. The sum across these values is always 1.

```
0         1
Date
2000-01-03  0.029665  0.970335
2000-01-04  0.000002  0.999998
2000-01-05  0.003385  0.996615
2000-01-06  0.003566  0.996434
2000-01-07  0.000665  0.999335
```

To further visualize the output we can plot smoothed regime probabilities. Now we can see our regime classifications taking shape, with our crises dates of interest having dominant probabilities of high variance regime returns.



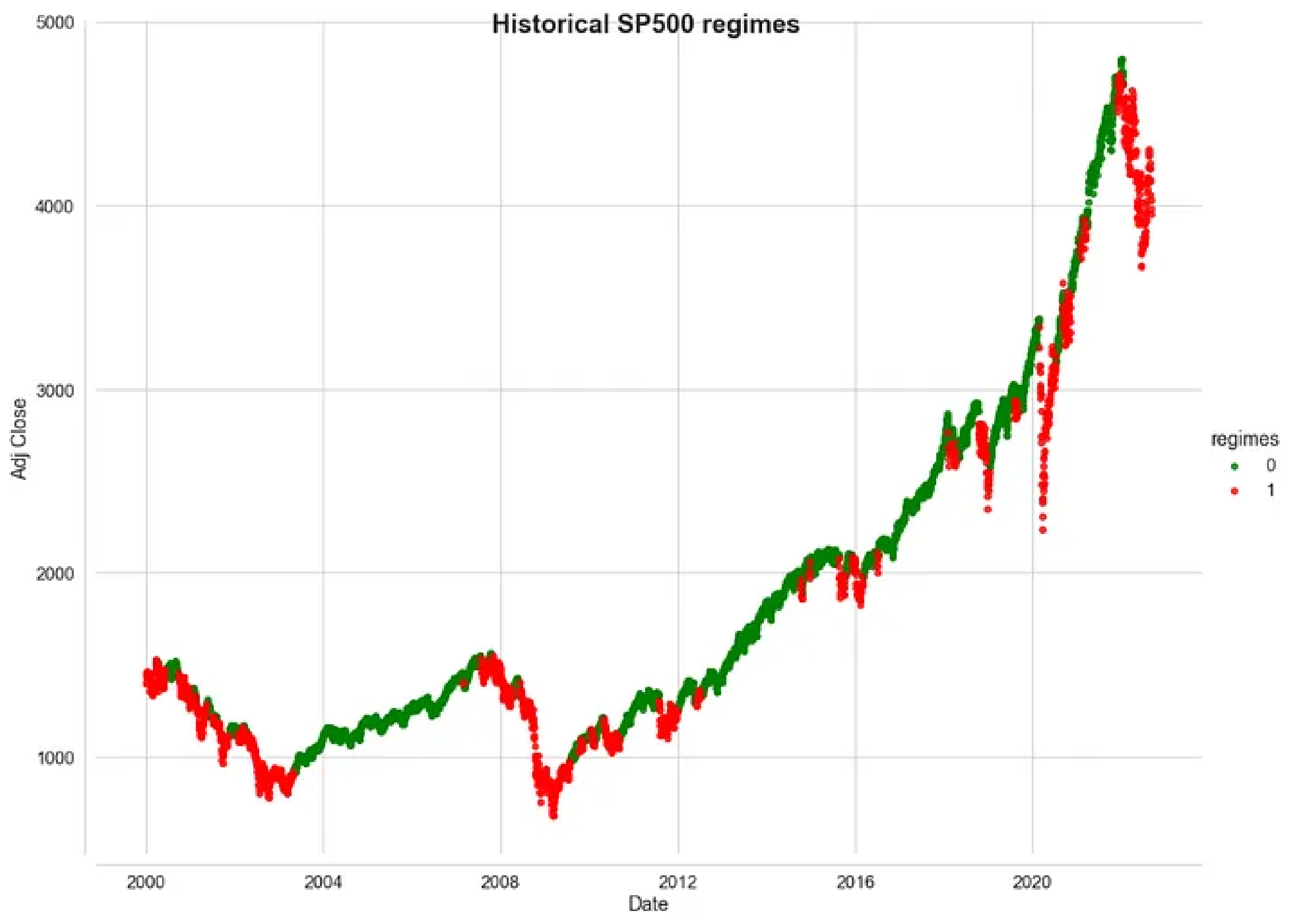
Although having probabilistic outputs is useful, it doesn't help us when we want to plot the binary classifications of our model. We create a binary output by encoding each regime as a 0 for low variance and 1 for high variance.

```python
low_var = list(res_kns.smoothed_marginal_probabilities[0])
high_var = list(res_kns.smoothed_marginal_probabilities[1])

regime_list = []
for i in range(0, len(low_var)):
    if low_var[i] > high_var[i]:
        regime_list.append(0)
    else:
        regime_list.append(1)
```

regime_encode_plot.py hosted with ❤ by GitHub view raw

Now all we have to do is visualize our regimes on a graph of the adjusted close price of the S&P 500.



What we get makes a lot of sense. During periods of volatility, generally around major market decline, we see high variance regime returns (red) confirming our hypothesis. During periods of low volatility, generally during calm uptrends we see low variance regime returns (green). From an investors perspective this graph can be interpreted simply, buy or hold assets during green periods and sell holdings as soon as possible in red periods. This is where machine learning may prove to be useful. A Markov switching regression is good at telling us what regime we were just in, 'nowcasting', but it doesn't not necessarily give us much information about the next regime. A machine learning model can provide just that, giving us an expectation of the next regime.

Random Forest Classifier

Machine learning is separated into three distinct types of learning, namely supervised, unsupervised and reinforcement learning. In this case we are going to be using a Random Forest Classifier, an algorithm in the supervised domain. Supervised learning implies that our model will take dependent and independent variables. Our dependent variables are our data labels which in this case will be the regime in the next period. Our independent variables are all the features which the model will use in making these predictions. At a high level, our machine learning model will aim to map a function between our labels and features. Our biggest challenge in applying machine learning to our problem is the relatively small dataset increasing the risk of overfitting. Overfitting means that our model memorizes our past data, learning few underlying relationships, which severely limits its usefulness looking forward.

A Random Forest is an estimator that fits a number of decision tree classifiers on various dataset subsamples using averaging to improve predictions. A single decision tree is a chart describing the process of reaching classification or regression decisions. In this case we are using a classification decision process.

First we need to prepare our dataset for the Random Forest Classifier. In addition to S&P 500 price data, we will create price and volume return features. These will be the percentage changes in the adjusted close price and volume over a number of different lookback periods. Most importantly we also need to append our MS-AR regime classification probabilities. These will describe the classification of the current regime to the machine learning model. We will also add return volatility statistics to describe the volatility of past prices. Finally we need our binary state or regime classifications, this is the column that we will transform into our labels for the classifier.

```python
ml_df = yf.download('^GSPC', start="2000-01-01", end="2022-09-01", interval="1d")

# price and volume returns
for i in [1, 2, 3, 5, 7, 14, 21]:
    ml_df[f'Close_{i}_Value'] = ml_df['Adj Close'].pct_change(i)
    ml_df[f'Volume_{i}_Value'] = ml_df['Volume'].pct_change(i)
ml_df.dropna(inplace=True)

# probabilities
low_var_prob = list(res_kns.smoothed_marginal_probabilities[0])
high_var_prob = list(res_kns.smoothed_marginal_probabilities[1])
ml_df['Low_Var_Prob'] = low_var_prob[len(low_var_prob)-len(ml_df):] # adjust length
ml_df['High_Var_Prob'] = high_var_prob[len(high_var_prob)-len(ml_df):]

# volatility
for i in [3, 7, 14, 21]:
    ml_df[f'Volt_{i}_Value'] = np.log(1 + ml_df['Close_1_Value']).rolling(i).std()

ml_df.dropna(inplace=True)

# states
ml_df['regimes'] = regime_list[len(regimes)-len(ml_df):] # adjust length

ml_df.head()
```

rf_data_prep.py hosted with ❤ by GitHub view raw

Labeling data in supervised learning is an important step and sometimes a bottleneck in the pipeline. This is where we describe the problem to the model. It is important to ask ourselves what we want from the model in practice. In this case, all we want is the model to tell us what it expects the next day's regime to be. Our model's target is going to be the binary classification of the next regime. We already have the binary classification of the current regime in the regimes column, so all we have to do is shift this column's values backwards. This brings the future regime into the present, providing us with our target y value.

```python
ml_df['regimes'] = ml_df['regimes'].shift(-1)
ml_df.dropna(inplace=True)
ml_df
```

rf_labels.py hosted with ❤ by GitHub view raw

Now that our regimes column is our target, we can start our data preprocessing. I will be using the train-test split function to create X and y lists for both training and testing. I opted for an 80–20 train-test split with 80% of the data used for training and 20% reserved for evaluating the model.

```python
X_train, X_test, y_train, y_test = train_test_split(rf_df, labels, test_size=0.2)
Counter(y_train)
```

train_test.py hosted with ❤ by GitHub view raw

Upon inspection we find that we have significantly fewer high variance regime training samples when compared with low variance regime training samples.

```
{0: 3113, 1: 1416}
```

This makes sense as times of volatility and instability are thankfully relatively infrequent, however this causes our dataset to be unbalanced. Unbalanced data means that our classes for classification are unequally represented which may affect model performance. To rebalance our dataset we have a few options. We could oversample our smallest class by synthetically creating more examples of high variance regimes. I prefer to take the more intuitive approach of under sampling the largest class by the size difference to the smallest class. This means that we will just get rid of low variance examples until our number of high and low variance regime return examples are equal.

```python
under_sampler = RandomUnderSampler(random_state=40)
X_rs, y_rs = under_sampler.fit_resample(X_train, y_train)
```

resample.py hosted with ❤ by GitHub view raw

Now that our data is ready we can fit our model. This is the easy part as all complex calculations and iterations are taken care of for us.

```python
rf = RandomForestClassifier(n_estimators=20)
rf.fit(X_rs, y_rs)
```

rf_fit.py hosted with ❤ by GitHub view raw

In fitting our model we need to specify the number of estimators which is the number of trees in our random forest. I arbitrarily chose to initialize the model with 20 trees. We will explore how to better choose this parameter later on in the article.

After fitting our model we can make predictions on our testing dataset which we will use to evaluate performance. We can get two types of predictive output from the random forest classifier. We can get a binary classification output or a probabilistic classification output.

```
y_pred = rf.predict(X_test)
y_prob_pred = rf.predict_proba(X_test)
```

rf_pred.py hosted with ❤ by GitHub view raw

To evaluate performance, we are going to use the receiver operating characteristic curve (ROC) function to retrieve the false positive and true positive rate. A ROC curve gives us the diagnostic ability of a binary classifier. It represents the trade-off between sensitivity (True positive rate) and specificity (1 — False positive rate).

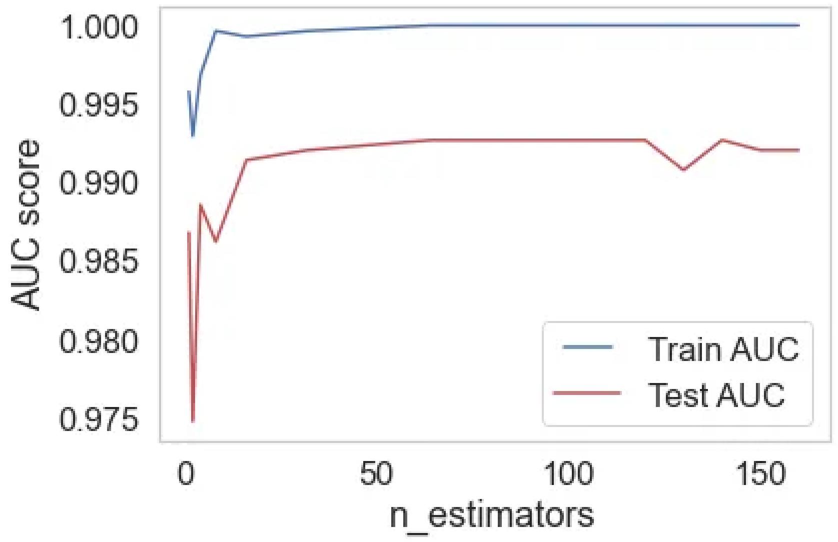
```
acc_score = accuracy_score(y_test, y_pred)
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
```

rf_performance.py hosted with ❤ by GitHub view raw

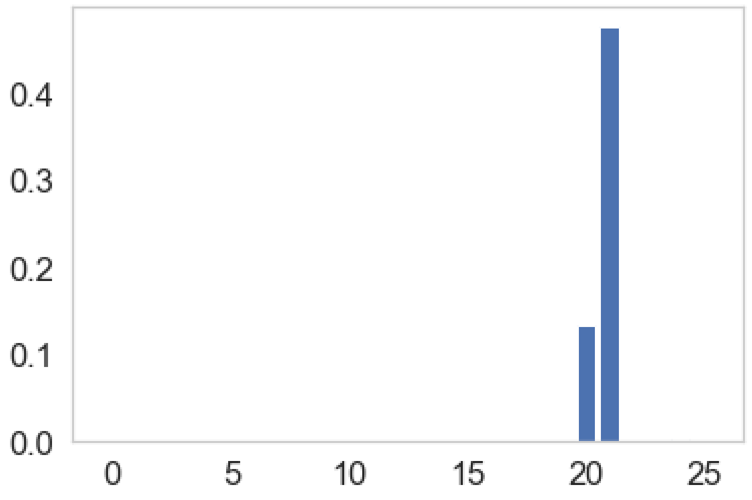
```
accuracy_score= 0.9947043248014121 roc_auc= 0.9945982714468629 FPR=
0.005121638924455826 TPR= 0.9943181818181818
```

On the testing set (unseen data), our model has achieved a true positive rate of 99.4% meaning that over 99% of actual high variance regimes in the next period were correctly predicted as high variance regimes. The model presented with a 0.5% false positive rate, meaning that 0.5% of actual low variance regimes in the next period were incorrectly predicted as high variance regimes. To get a view of overall accuracy we can compute the area under the curve (AUC). Our model has an AUC of 99.5%, meaning that the model is able to distinguish between high and low variance regimes 99.5% of the time. Alternatively we can also use an accuracy score computation. All metrics would suggest a model with a high degree of accuracy and low prediction error, providing a satisfactory result for this use case.

Previously I mentioned that I had arbitrarily chosen to initialize the random forest with 20 trees. Now this has seemed to work relatively well, but how would we go about tuning that hyperparameter? Although tuning would likely yield a small increase in results, it is important to remember the random nature of machine learning and that every model fitment is going to contain some variation. To tune a hyperparameter such as the number of trees we can easily write a loop to fit models of different specifications and plot the results. Visually we can see that in this case there is not too much extra performance to gain from increasing the number of trees beyond approximately 20 trees.



In addition to hyperparameter tuning we can also improve our models by providing more useful or important features. We can do this by checking which features the model deems as important. To check for the importance of a feature to our particular model we will use a permutation importance check. Permutation importance differs from traditional feature importance in that instead of reflecting the intrinsic predictive value of a feature itself, it reflects how important a particular feature is to our chosen model. In our current case, it makes more sense using permutation importance as we just want an idea of which features are contributing to model performance, however if we were wanting to understand whether certain features held predictive value then feature importance would be the better choice.



As one would expect our most important features are columns 20 and 21, being our current variance regime probability columns. It would seem that our model has learnt rules associating the current regime mapped to the future regime.

. . .

This wraps up our exploration of using machine learning to predict the next variance return regime. I have two areas that I would target when building on and improving this approach.

1. The addition of a bullish (uptrend) and bearish (downtrend) indication to state labels. This can be done by adding a moving average to price and classifying states into not only high and low variance returns but also bullish when price is above the moving average and bearish when below. This will yield four regimes for classification: bearish low variance, bullish low variance, bearish high variance and bullish high variance. A four state classification drastically reduces the number of examples, increasing the overfitting difficulties when applying machine learning. A solution to this problem would be to apply these methods to a time series of higher granularity like hourly price data.
2. Using deep learning for predicting the next state. Leveraging long short-term memory neural networks could allow for a comprehensive predictive model that can detect non-linear dependencies within the series over a specified lookback window. Again the amount of data is the limiting factor in this application, making it more useful for higher granularity time series.

I hope you found this simple introduction to regime analysis and prediction useful. My takeaway from this process is the different angle of approach to time series problems. While many may try to better predict a series, an approach like this suggests that it may be better to know when to predict and when not to predict. In terms of finance, this translates to analyzing when to trade and when not to trade a particular strategy. Find the full notebook and code for this article here.

. . .

[1] https://en.wikipedia.org/wiki/Autoregressive_model

[2] https://en.wikipedia.org/wiki/Markov_chain

[3] Mendy, D. & Widodo, T. (2018). ”Two stage Markov switching model: Identifying the Indonesian Rupiah Per US Dollar turning points post 1997 financial crisis,” MPRA Paper 86728.

Data Science Time Series Analysis Finance

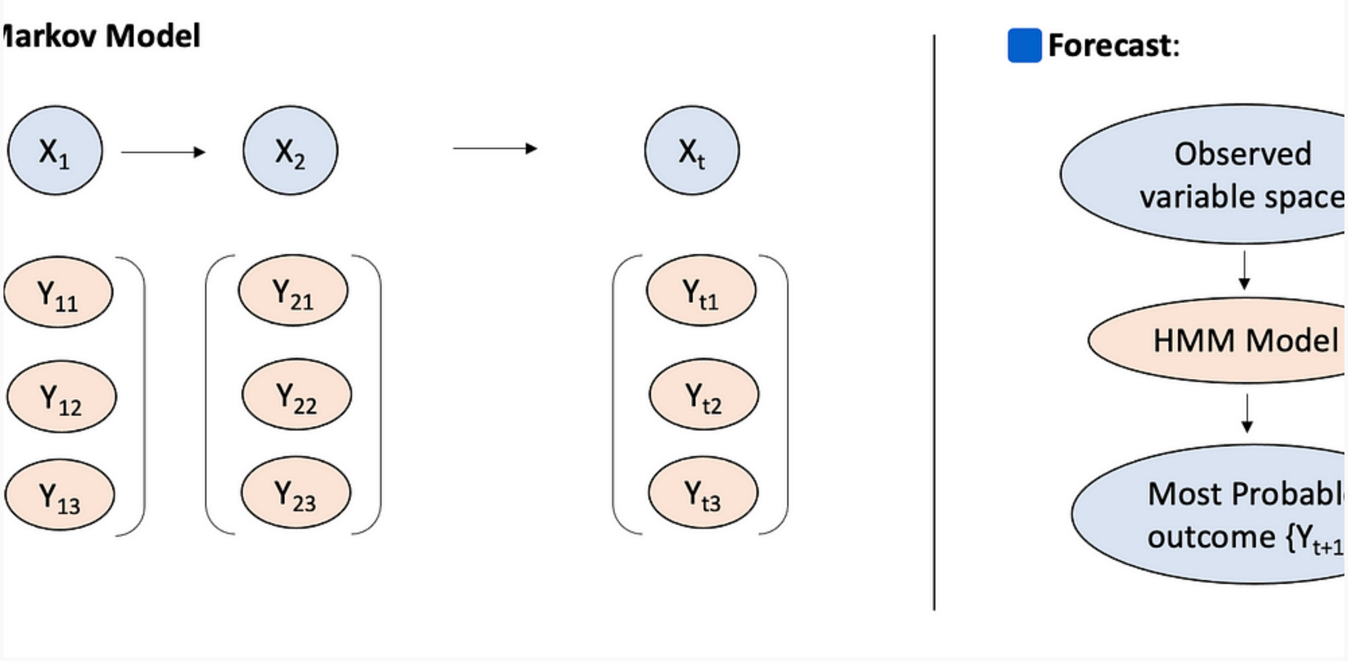
S

Follow

Written by Spencer

73 Followers

Recommended from Medium



ASHISH DHIMAN in DataDrivenInvestor

Time Series forecasting in Python with Hidden Markov Models

This article focuses on how to use HMM not only to model the underlying process but further use it to make out of sample forecasts.

8 min read · Jun 9

36

The Data Beast

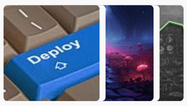
Time Series :: Part-8 :: SARIMA Model

The SARIMA (Seasonal Autoregressive Integrated Moving Average) model is an extension of the ARIMA model that is used to model time series...

3 min read · Apr 30

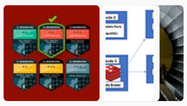
5

Lists



Predictive Modeling w/ Python

20 stories · 513 saves



New_Reading_List

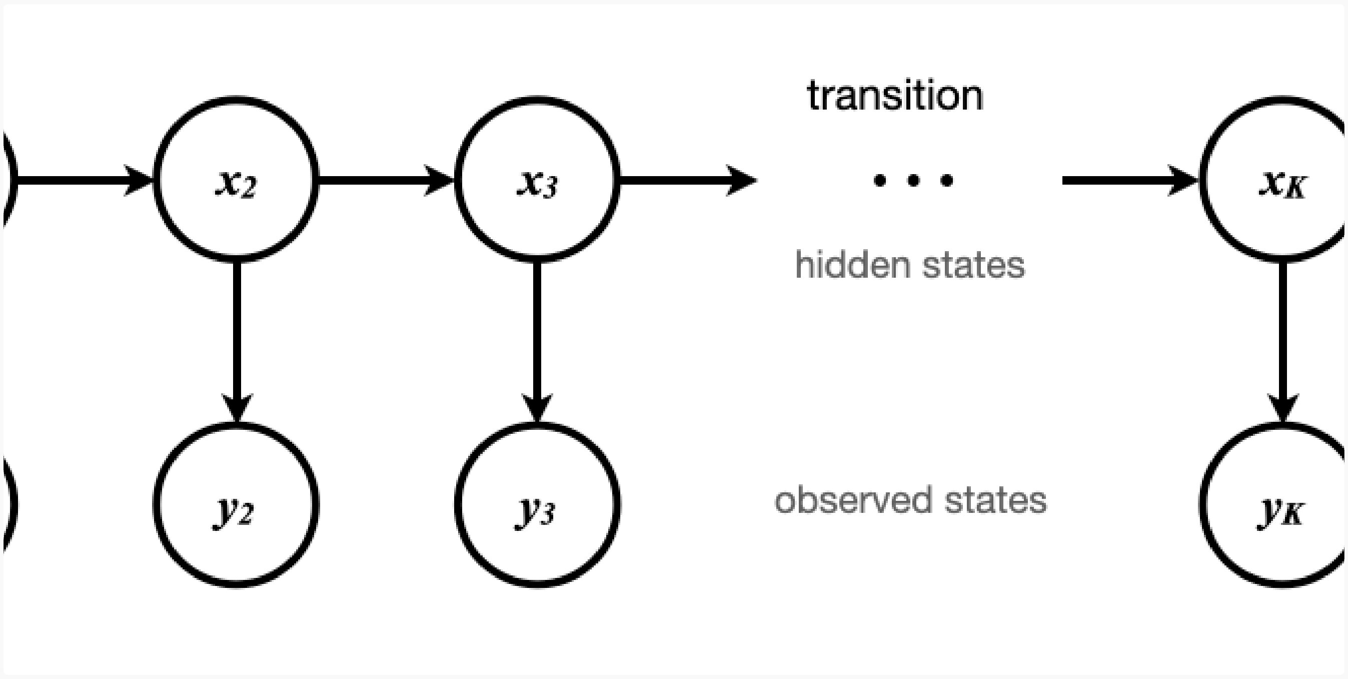
174 stories · 152 saves

Practical Guides to Machine Learning

10 stories · 585 saves

Coding & Development

11 stories · 226 saves



$$\begin{tikzcd} \cdots \arrow[r] & x_2 \arrow[r] \arrow[d] & x_3 \arrow[r] \arrow[d] & \cdots \arrow[r] & x_K \arrow[d] \\ \cdots & y_2 & y_3 & & y_K \end{tikzcd}$$

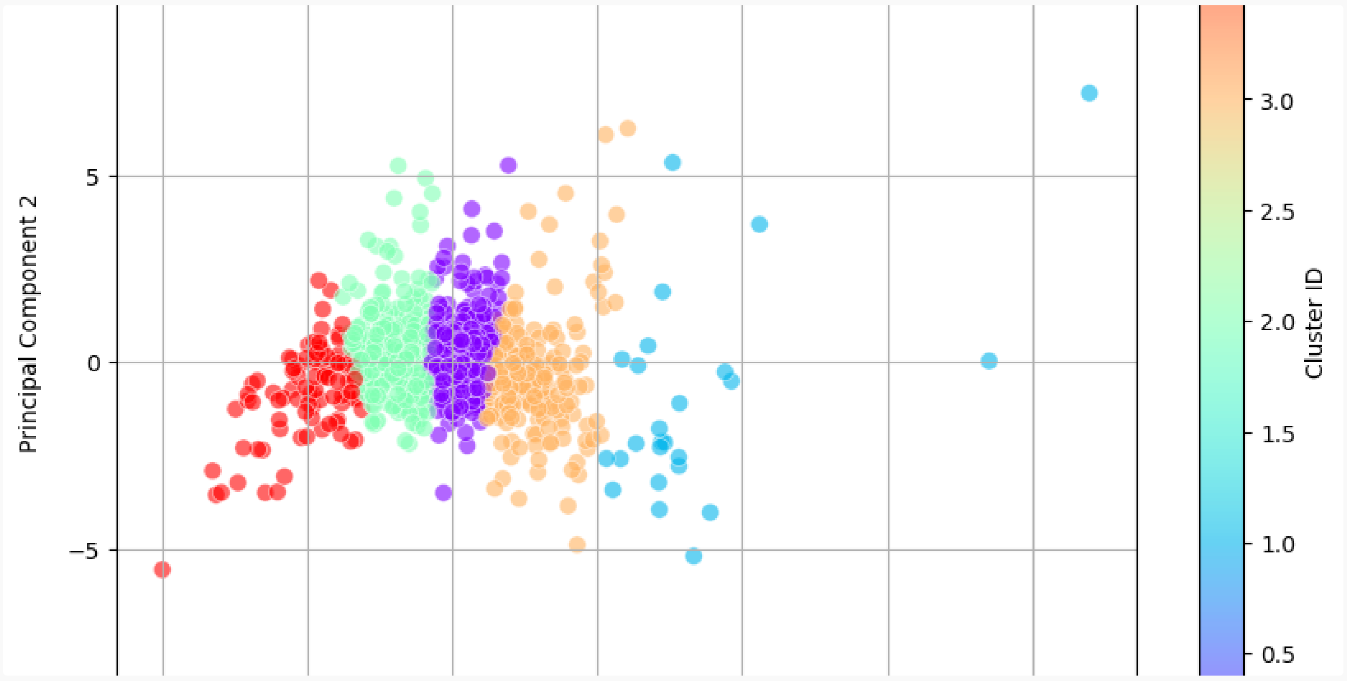
Ellie Arbab

Hidden Markov Models

Viterbi Algorithm

6 min read · Jun 1

10



S.M. Laignel in InsiderFinance Wire

Back to stocks clustering: the coded approach

How to cluster a universe of stocks using k-means algorithm in Python and raw data extracted from FinViz.

8 min read · Oct 8

6

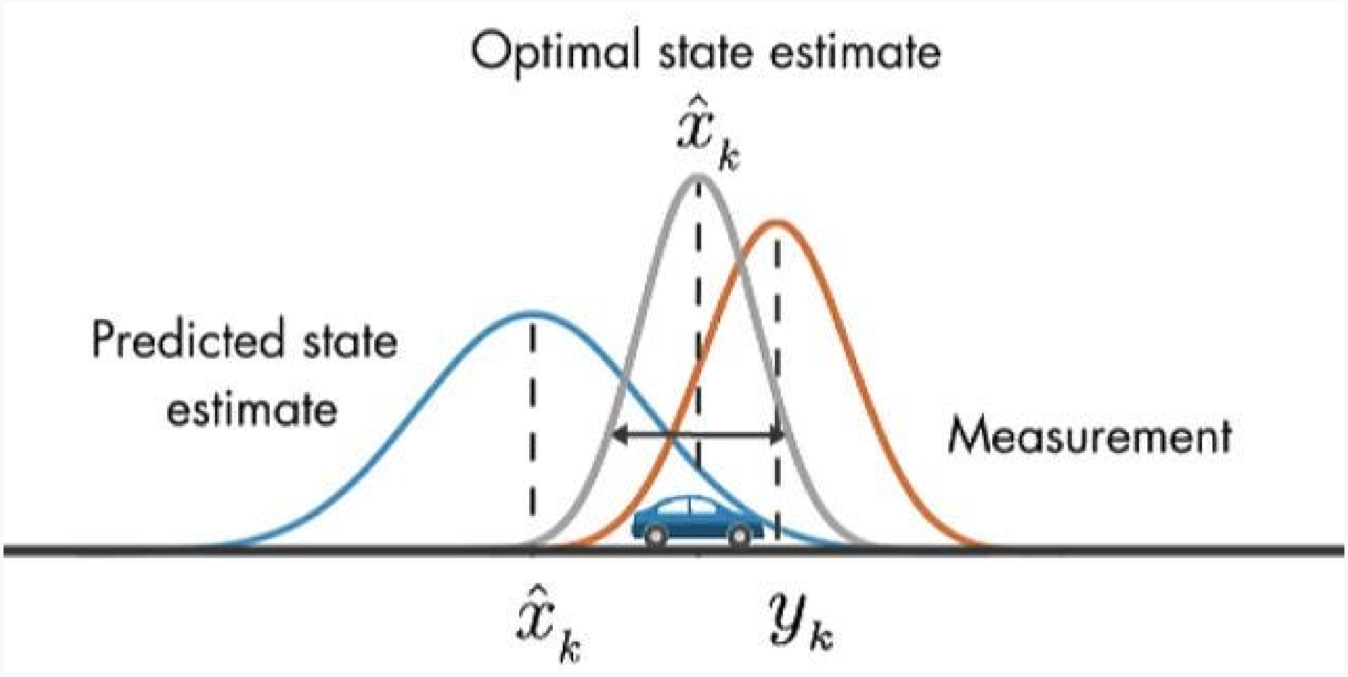
Diego Pesco Alcalde

The Journey of a Quantitative Trading Strategy Development (Part 5)

A no-code overview

6 min read · Jun 4

20



Dhanoop Karunakaran in Intro to Artificial Intelligence

Kalman filter in stock trading

What is Kalman Filter?

5 min read · Sep 11

103

See more recommendations