Machine Learning – Regime Switching HMM
Phase II

November 2021

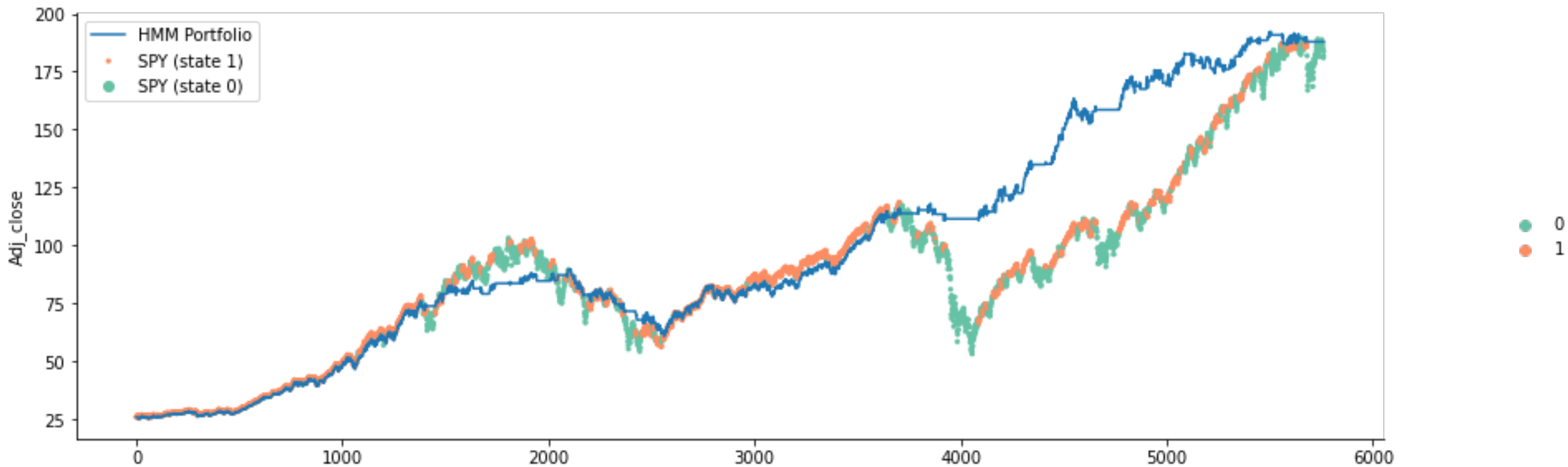
By: Corbin Apple & Young Seok Seo

Table of Contents

Project Summary

Summary

- **Main Idea:** Take advantage of changing market dynamics
- **Model:** Hidden Markov Model, identifying stable, low variance regimes and unstable, high variance regimes
- **Idealized Results:** Construct a replicating portfolio that avoids market drawdowns, but capitalizes on positive performance



Model Overview

Overview

- **Our Idea:**
  1. Fit a Hidden Markov Model using a rolling window
  2. Estimate the current state we are in at each time step, using only historical data
  3. Construct a portfolio in which we are fully invested if we are currently in a positive market, and fully divested in a negative market
- **Reasoning:**
  - Predicting future daily returns is very challenging
  - This method allows for model construction purely on actual returns
  - Ease of implementation and reproducibility

Code Breakdown

Breakdown (1 of 11)

- Import necessary packages

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import datetime
from hmmlearn import hmm

import warnings
warnings.filterwarnings("ignore")
```

Code Breakdown (cont.)

Breakdown (2 of 11)

- Import Bank of America historical data

```python
input_df = pd.read_csv('~/Desktop/Corbin SBU/AMS 520/Project/BofA Projects Data/EOD_20210908.csv',
                       header = None,
                       names = ['Ticker', # Label columns
                                'Date',
                                'Open',
                                'High',
                                'Low',
                                'Close',
                                'Volume',
                                'Dividend',
                                'Stock_split',
                                'Adj_open',
                                'Adj_high',
                                'Adj_low',
                                'Adj_close',
                                'Adj_volume'])
```

Code Breakdown (cont.)

Breakdown (3 of 11)

- Select chosen index (SPY)
- Calculate daily percentage returns
- Calculate volatility of daily returns over selected time window (500 days)

```
# Proposed Idea: Create a HMM for the recent Neff days, and for all days after the Neff'th day
# Predict which state we are currently in based on the Neff recent days
# Be fully invested if in a positive market, fully divested in a negative market

data = input_df.loc[input_df['Ticker'] == 'SPY'] # Select which index to use for analysis
data.reset_index(inplace=True, drop=True)

Neff = 500 # Lookback length

# Calculate daily percentage returns
Return = 100*(data['Adj_close'] - data['Adj_close'].shift(1)) / data['Adj_close'].shift(1)

data['Return'] = Return
```

```
# Volatility = standard deviation for trailing Neff days
Volatility = np.concatenate((np.zeros(Neff),
                             [np.sqrt(sum((data['Return'][i:Neff+i] -
                                           np.mean(data['Return'][i:Neff+i]))**2) / Neff)
                              for i in range(len(data)-Neff)]))

data['Volatility'] = Volatility
```

Data Analysis

Data Details

	Ticker	Date	Open	High	Low	Close	Volume	Dividend	Stock_split	Adj_open	Adj_high	Adj_low	Adj_close	Adj_volume	Return	Volatility
0	SPY	1993-01-29	43.9687	43.9687	43.7500	43.9375	1003200.0	0.0	1.0	25.804239	25.804239	25.675889	25.785928	1003200.0	NaN	0.000000
1	SPY	1993-02-01	43.9687	44.2500	43.9687	44.2500	480500.0	0.0	1.0	25.804239	25.969328	25.804239	25.969328	480500.0	0.711238	0.000000
2	SPY	1993-02-02	44.2187	44.3750	44.1250	44.3437	201300.0	0.0	1.0	25.950958	26.042687	25.895968	26.024318	201300.0	0.211751	0.000000
3	SPY	1993-02-03	44.4062	44.8437	44.3750	44.8125	529400.0	0.0	1.0	26.060998	26.317757	26.042687	26.299446	529400.0	1.057196	0.000000
4	SPY	1993-02-04	44.9687	45.0937	44.4687	45.0000	531500.0	0.0	1.0	26.391117	26.464476	26.097678	26.409486	531500.0	0.418410	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7199	SPY	2021-08-31	452.1300	452.4900	450.9200	451.5600	58631140.0	0.0	1.0	452.130000	452.490000	450.920000	451.560000	58631140.0	-0.148155	1.577472
7200	SPY	2021-09-01	452.5600	453.1100	451.5450	451.8000	48667698.0	0.0	1.0	452.560000	453.110000	451.545000	451.800000	48667698.0	0.053149	1.577512
7201	SPY	2021-09-02	453.3200	454.0500	451.9100	453.1900	42479834.0	0.0	1.0	453.320000	454.050000	451.910000	453.190000	42479834.0	0.307658	1.577511
7202	SPY	2021-09-03	451.9800	453.6300	451.5500	453.0800	47155405.0	0.0	1.0	451.980000	453.630000	451.550000	453.080000	47155405.0	-0.024272	1.577528
7203	SPY	2021-09-07	452.7100	452.8100	450.7423	451.4600	51477698.0	0.0	1.0	452.710000	452.810000	450.742300	451.460000	51477698.0	-0.357553	1.577303

7204 rows × 16 columns

Code Breakdown (cont.)

Breakdown (4 of 11)

- Fit the Hidden Markov Models and predict the regime given the current observations

```
# Initialize a HMM
states = 2
max_iterations = 100 # For EM algorithm
current_state = np.array([]) # Initialize an array to track the current regimes

# Exclude first (to calculate trailing volatility) and
# second (to have a full set of observations to fit a HMM) Neff observations
for i in range(1, len(data) - 2*Neff):
    # Initialize a Gaussian HMM
    model = hmm.GaussianHMM(n_components = states, covariance_type="full", n_iter = max_iterations);
    # Pull Neff observations of Volatility and Returns
    observations = np.stack((data['Volatility'][Neff+i:Neff*2+i], data['Return'][Neff+i:Neff*2+i]), axis=1)
    model.fit(observations) # Fit the model to the observations
    print(f'i = {i}') # Print to ensure loop is running
    # Model randomly allocates a '0' or a '1' to a state, so check which state has the higher mean return
    if model.means_[0,1] > model.means_[1,1]:
        positive_state = 0
    else:
        positive_state = 1
    predictions = model.predict(observations) # Predict the state for each observation
    if positive_state == 1: # If the state with the higer mean return is state 1, do nothing, if not...
        pass
    else: # Switch the predicted regimes to ensure state 1 is always the regime with a greater mean return
        zeros = np.where(predictions == 0)
        ones = np.where(predictions == 1)
        predictions[zeros] = 1
        predictions[ones] = 0
    # Append the current state to the regime tracker
    current_state = np.append(current_state,predictions[-1])
# Switch values from type float to type int for later calculations
current_state = current_state.astype(np.int64)
```

Code Breakdown (cont.)

Breakdown (5 of 11)

- Initial review of the regime switching frequency

```
total_regime_switches = sum(np.abs(current_state[1:] - current_state[:-1]))
years_of_data = len(current_state) / 252 # Approximately 252 trading days in a year
avg_regime_changes_per_year = total_regime_switches / years_of_data

print(f'Total regime switches: {total_regime_switches}')
print(f'Average number of regime switches per year: {avg_regime_changes_per_year:.01f}')
```

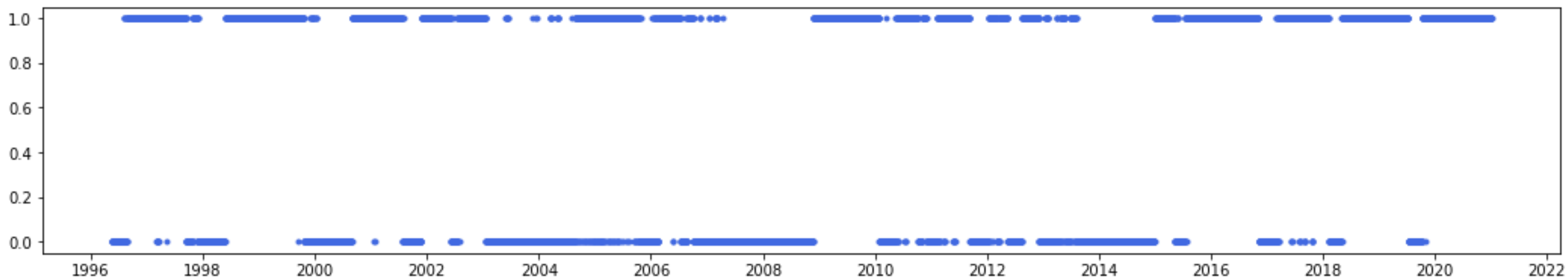
```
Total regime switches: 349
Average number of regime switches per year: 14.2
```

Code Breakdown (cont.)

Breakdown (6 of 11)

- Visually consider the projected regime switching

```
# Plot regime switches to graphically analyze

dates = [datetime.datetime.strptime(d,"%Y-%m-%d").date() for d in data.Date[2*Neff + 1:]]

fig, ax = plt.subplots(figsize=(18,4))

formatter = mdates.DateFormatter("%Y")

ax.xaxis.set_major_formatter(formatter)

fmt_half_year = mdates.MonthLocator(interval=24)
ax.xaxis.set_major_locator(fmt_half_year)

ax.plot(dates, current_state, '.', color = 'royalblue');
```



Code Breakdown (cont.)

Breakdown (7 of 11)

- Numerically analyze the mean return and volatility of the two states

```
state_0 = np.where(current_state == 0)[0] + 2*Neff + 1
state_1 = np.where(current_state == 1)[0] + 2*Neff + 1
```

```
# Calculate and view statistics of the two states

print(f'Number of occurences of state 0: {len(state_0)}')
print(f'Mean return of state 0: {data.loc[state_0].Return.mean():.03f}')
print(f'Volatility of state 0: {data.loc[state_0].Return.std():.03f}')
print('\n')
print(f'Number of occurences of state 1: {len(state_1)}')
print(f'Mean return of state 1: {data.loc[state_1].Return.mean():.03f}')
print(f'Volatility of state 1: {data.loc[state_1].Return.std():.03f}')
```

```
Number of occurences of state 0: 2708
Mean return of state 0: 0.050
Volatility of state 0: 1.469


Number of occurences of state 1: 3495
Mean return of state 1: 0.038
Volatility of state 1: 1.040
```

Code Breakdown (cont.)

Breakdown (8 of 11)

- Plot the HMM portfolio against the actual results

```
# Calculate the growth of a theoretical portfolio
# Assume fully invested if state 1, fully divested if state 0

current_portfolio = data.Adj_close[2*Neff+1] * np.cumprod(current_state * data.Return[2*Neff+1:] / 100 + 1)
```

```
# Plot the return portfolio dynamics

plot = sns.relplot(x = range(0,len(current_state)),
                   y = "Adj_close",
                   data = data[2*Neff+1:],
                   hue = current_state,
                   linewidth = 0,
                   palette = "Set2",
                   s = 10);

plt.plot(range(0,len(current_state)), current_portfolio, color='royalblue')

plot.fig.set_size_inches(18,10)
```

Code Breakdown (cont.)

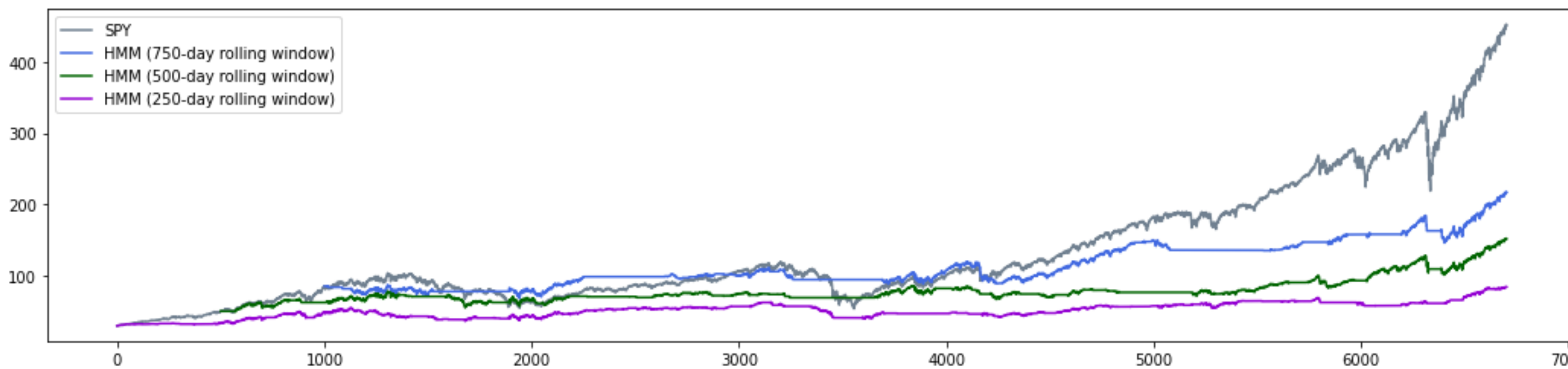
Breakdown (9 of 11)

Code Breakdown (cont.)

Breakdown (10 of 11)

- Consider alternative rolling windows

```
# Import previous iterations using different Neff window lengths
test_portfolio_750 = np.loadtxt("test_portfolio_750.csv")
test_portfolio_500 = np.loadtxt("test_portfolio_500.csv")
test_portfolio_250 = np.loadtxt("test_portfolio_250.csv")

fig, ax = plt.subplots(figsize=(18,4))

ax.plot(range(0,len(current_state)+500), data.Adj_close[2*250+1:], color='slategray');
ax.plot(range(1000,len(current_state)+500), test_portfolio_750, color='royalblue');
ax.plot(range(500,len(current_state)+500), test_portfolio_500, color='darkgreen');
ax.plot(range(0,len(current_state)+500), test_portfolio_250, color='darkviolet');

ax.legend(['SPY','HMM (750-day rolling window)', 'HMM (500-day rolling window)', 'HMM (250-day rolling window)']);
```



Code Breakdown (cont.)

Breakdown (11 of 11)

- Calculate empirical results

```python
# Empirical results

spy_sharpe = data.Return[2*Neff+1:].mean() / data.Return[2*Neff+1:].std()

test_portfolio_750_returns = (test_portfolio_750[1:] - test_portfolio_750[:-1]) / test_portfolio_750[:-1]
test_portfolio_750_sharpe = test_portfolio_750_returns.mean()/test_portfolio_750_returns.std()

test_portfolio_500_returns = (test_portfolio_500[1:] - test_portfolio_500[:-1]) / test_portfolio_500[:-1]
test_portfolio_500_sharpe = test_portfolio_500_returns.mean()/test_portfolio_500_returns.std()

test_portfolio_250_returns = (test_portfolio_250[1:] - test_portfolio_250[:-1]) / test_portfolio_250[:-1]
test_portfolio_250_sharpe = test_portfolio_250_returns.mean()/test_portfolio_250_returns.std()

print(f'SPY average return: {spy_mean_return:.03f}')
print(f'SPY volatility: {spy_vol:.03f}')
print(f'SPY Sharpe Ratio: {spy_sharpe:.03f}')
print('\n')
print(f'HMM (750-day rolling window) average return: {data.Return[2*Neff+1:].mean() :.05f}')
print(f'HMM (750-day rolling window) volatility: {data.Return[2*Neff+1:].std():.05f}')
print(f'HMM (750-day rolling window) Sharpe Ratio: {test_portfolio_750_sharpe:.03f}')
print('\n')
print(f'HMM (500-day rolling window) average return: {test_portfolio_500_returns.mean():.05f}')
print(f'HMM (500-day rolling window) volatility: {test_portfolio_500_returns.std():.05f}')
print(f'HMM (500-day rolling window) Sharpe Ratio: {test_portfolio_500_sharpe:.03f}')
print('\n')
print(f'HMM (250-day rolling window) average return: {test_portfolio_250_returns.mean():.05f}')
print(f'HMM (250-day rolling window) volatility: {test_portfolio_250_returns.std():.05f}')
print(f'HMM (250-day rolling window) Sharpe Ratio: {test_portfolio_250_sharpe:.03f}')
```

Empirical Results

Initial Results

```
SPY average return: 0.043
SPY volatility: 1.246
SPY Sharpe Ratio: 0.035


HMM (750-day rolling window) average return: 0.04337
HMM (750-day rolling window) volatility: 1.24552
HMM (750-day rolling window) Sharpe Ratio: 0.026


HMM (500-day rolling window) average return: 0.00021
HMM (500-day rolling window) volatility: 0.00781
HMM (500-day rolling window) Sharpe Ratio: 0.027


HMM (250-day rolling window) average return: 0.00019
HMM (250-day rolling window) volatility: 0.00747
HMM (250-day rolling window) Sharpe Ratio: 0.025
```

Sharpe Ratios: SPY > HMM (500) > HMM (750) > HMM (250)

Next Steps

Future Ideas for Phase III

- Performing this same analysis with adjusted factors, including using:
  - Different rolling window lengths (e.g. 10-1,000 days), in search of an "optimal" lookback timeframe
  - Other indices (domestic and international), to see if an "optimal" window length is consistent across funds and markets
  - Incorporating transaction costs each time we switch regimes, with the goal of penalizing models that recommend frequent rebalancings
- Should we decide to go down the route of predicting day-ahead returns, incorporating these forecasts into out Hidden Markov Model, we will likely try to implement a standard time series model (e.g. GARCH, ARMA-GARCH)
  - Predict future states by fitting forecasted observations
  - Make optimal investment decisions in advance of realized returns
- Reorganize main code as a single function to support easy reproducibility of results
- Produce results with visualization package (based on Shiny framework, QuantStats)

References

Reference List

- Peter Nystrup, Henrik Madsen & Erik Lindström (2018) Dynamic portfolio optimization across hidden market regimes, Quantitative Finance, 18:1, 83-95, DOI: 10.1080/14697688.2017.1342857
- Wang M, Lin Y-H, Mikhelson I. Regime-Switching Factor Investing with Hidden Markov Models. Journal of Risk and Financial Management. 2020; 13(12):311. https://doi.org/10.3390/jrfm13120311