Market Regime Detection and Portfolio Optimization

Using a Hidden Markov Model and Markowitz Mean-Variance Portfolio Theory

Vadim Bodnarenco

September 2025

Chapter 1

Abstract

This project explores how different market regimes affect the optimal weights in portfolios of common assets. A Hidden Markov Model (HMM) was implemented to identify market regimes and classify them as either bull or bear based on the log returns of a global equity index ETF. A Monte Carlo simulation was then used to generate random portfolio weights and visualize them on a risk–return plot. The Markowitz mean–variance optimization was subsequently applied to find the efficient frontier and the optimal portfolio weights corresponding to the maximum Sharpe ratio. The results show that during bear markets, the algorithm tends to allocate higher weights to safer assets, while during bull markets it favors assets with higher expected returns. These findings demonstrate how statistical regime detection can be effectively combined with classical portfolio theory to improve asset allocation and risk management.

Contents

Chapter 2

Introduction

It is widely recognized that optimal portfolio composition can vary significantly under different market conditions. During so-called bull markets, characterized by rising prices and positive investor sentiment, assets such as equities, exchange-traded funds, and cryptocurrencies tend to deliver higher returns. Conversely, in bear markets, when prices fall and volatility rises, these same assets often experience lower or negative returns, while more defensive assets such as government bonds may perform relatively better.

This report is aimed at investigating the optimal allocation of common assets during these different market regimes. To do that, a Hidden Markov Model is implemented to identify whether the market is in a bull or bear regime. For each regime, portfolio optimization is performed using the Markowitz mean-variance portfolio optimization method, supported by Monte Carlo simulations to visualize the investment set, efficient frontier, and the most optimal Sharpe ratio.

Altogether, this can be summarized in a research question:

How do the risk–return characteristics and optimal portfolio weights differ between bull and bear markets as identified by a Hidden Markov Model of market returns?

By comparing the resulting optimal portfolios across different market regimes, this paper aims to help investors adjust their asset allocation strategies to achieve the most optimal return.

Chapter 3

Main Body

It is widely recognized that during so-called bull markets, riskier assets such as cryptocurrencies, common stocks, and options tend to generate higher average returns. During bear markets on the other hand, these riskier assets are often outperformed by less risky assets such as treasury bonds. One of the objectives of this project is to empirically confirm this pattern. Therefore, the following hypothesis is proposed: During bull markets, the portfolio weights of riskier assets will dominate, whereas during bear markets, the majority of risky assets will have weights close to zero, with less risky assets becoming dominant.

3.0.1 Data Overview

First of all, in order to some asset is needed that accurately captures the behavior of general market. An FTSE All World ETF has been chosen for this purpose, due to its broad exposure not just to the large-cap stocks but also to mid and smaller sized companies. A portfolio was chosen to be constructed with six different assets, of which three stocks from different countries for international exposure - Toyota from Japan, ASML from the Netherlands and Meta from the US. An ETF tracking the price of Bitcoin was chosen in order to contain an asset with higher than average volatility, as well as an already mentioned FTSE All World ETF for some diversification. Finally, a US 1-3 years treasury bond, as a theoretically safe asset during the bear market. Each of these assets' daily closing price has been downloaded from the Market Watch website [1], with a one year period, same for all the assets. These data have been organized into a time series and sorted in ascending order such that the newest data is in the tail of the series. Log-returns were then calculated, as they are time-additive, and annualized by multiplying them by the number of trading days (252).

Stocks	**Cryptocurrency**	**ETF**	**Bonds**
Meta ASML Toyota	BTC	FTSE All World ETF	US Gov. Bond (1–3y)

Table 3.1: Assets Used in the Portfolio

3.0.2 Market Regime Identification

This is the part where the actual coding starts. First of all, before thinking about asset allocation, a market regime needs to be identified. In order to do that, a Hidden Markov Model from the hmmlearn Python library has been implemented. It assumes that the log returns are represented by two hidden states: one with higher returns and lower variance (bull), and the other with lower returns and higher variance (bear).

```
model = hmm.GaussianHMM(n_components=2, covariance_type='diag', n_iter=1)
model.startprob_ = np.array([0.5, 0.5])
model.transmat_ = np.array([[0.98, 0.02], [0.05, 0.95]])
model.means_ = np.array([[bull_mean], [bear_mean]])
model.covars_ = np.array([[bull_var], [bear_var]])

Z = model.predict(X)
```

Listing 3.1: Hidden Markov Model for Market Regimes

The model initializes with two Gaussian distributions, with the starting probabilities of each being 0.5, so that at the start each of the states is equally likely. The matrix of transition probabilities is then created, with the assumption that the market tends to remain in a given state — hence a 98% chance that a bullish market remains bullish, and a 95% chance that a bearish market remains bearish. The model takes into consideration both the average return and the variance (volatility) to determine which regime a given return belongs to. The model then assigns each return to its respective state based on these parameters. A graph of market regimes is shown in 3.1, where the closing price has been plotted, with the bull market period highlighted in green and the bear period in red.

Figure 3.1: Visualization of the General Market With Bull and Bear States Highlighted in Green and Red, Respectively.

The figure makes sense, as it can be seen that when the model predicts a bear market, the corresponding price trend is generally decreasing, while during the bull state, the prices generally grow.

3.0.3 Portfolio Optimization

This part of the report explains the process of portfolio optimization. It is done in three steps. Firstly, a Monte Carlo simulation generates random weights for all six assets across many portfolios (200,000 in this case), constrained to add up to 1 in total. The code is written in a way that allows the user to choose whether to allow short selling or not. In the case where short selling is allowed, each weight can take a negative value, while the constraint mentioned above remains valid. For every portfolio, an expected return and risk (volatility) are calculated, which are then plotted on a risk–return graph. The next step is to generate an efficient frontier. This is done by solving the Markowitz Mean–Variance Optimization problem, which will be discussed later. The purpose of this step is to find a set of portfolios offering the least amount of risk for each level of return. Finally, an SLSQP optimizer is used to maximize the so-called Sharpe Ratio, which identifies the portfolio with the highest possible risk-adjusted return. This will also be discussed later. At the end, the code prints the asset weights corresponding to the maximum Sharpe ratio. This section is broken down into three parts to discuss each of these steps in detail.

Monte-Carlo Simulation

First of all, the Monte Carlo function takes as inputs the mean return vector containing the annualized returns for all of the assets, the covariance matrix of annualized returns for all of the assets, and the number of portfolios it needs to simulate. It then initiates a loop generating random weights subject to the following constraint:

$$\sum_{i=1}^{D} w_i = 1 \tag{3.1}$$

Where w_i is each asset's weight.

Next, risk (σ) and return (r) are calculated for each portfolio with $\boldsymbol{\mu}$ being a vector of expected returns for each asset, and w being the portfolio weights vector, using:

$$E[r_p] = w^T \boldsymbol{\mu} \tag{3.2}$$

and

$$\sigma_p^2 = w^T \sum w \tag{3.3}$$

Which correspond to the theoretical expressions for portfolio mean and variance presented in Section 3.2 of Financial Markets Theory [2]. These formulas are implemented in code, as shown in Listing 3.2 In addition to the main constraint, the bounds are defined such that the maximum short position for each individual asset is -0.5, while there is no upper bound. For long-only portfolios, these bounds are in the range (0, 1). The final outcome of the Monte Carlo function in this case are three arrays, which can be seen in the table 3.2.

Array Name	Dimensions	Description
`Rets`	(N,)	Expected return for each portfolio
`Risks`	(N,)	Portfolio volatility for each simulation
`Weights`	(N, D)	Matrix of portfolio weights

Table 3.2: Description of Arrays Returned From Monte Carlo Simulation, Where N is the Number of Simulated Portfolios and D is the Number of Assets.

The code for Monte Carlo simulation can be seen in Listing 3.2, while the code for the bounds is shown in Listing 3.3.

```
def Monte_Carlo_Short(mean_return, cov_np, N=1000000):
    all_returns = np.zeros(N)
    all_risks = np.zeros(N)
    weights = np.zeros((N, D))

    for i in range(N):
        rand_range = 1.0
        w = np.random.random(D)*rand_range - rand_range/2
        w[-1] = 1 - w[:-1].sum()
        np.random.shuffle(w)

        ret = mean_return.dot(w)
        risk = np.sqrt(w.dot(cov_np).dot(w))

        all_returns[i] = ret
        all_risks[i] = risk
        weights[i, :] = w

    return all_risks, all_returns, weights
```

Listing 3.2: Monte Carlo Function (Short Selling Allowed)

```
if position == 1:
    risks, rets, weights = Monte_Carlo_Short(mean_return,
       cov_np, N=200000)
    bounds = [(-0.5, None)] * D
else:
    risks, rets, weights = Monte_Carlo_Long(mean_return,
       cov_np, N=200000)
    bounds = [(0, 1)] * D
```

Listing 3.3: Bounds for Monte Carlo Simulation

Efficient Frontier

After the Monte Carlo simulation, a question arises: What is the minimum risk for each portfolio return? Barucci restates this more formally as 'We are interested in determining the portfolio $w^* \in \Delta_N$, which achieves the minimum variance among all possible portfolios with a given expected return $\mu \in R$ [2]. The solution to this problem obviously would be to minimize the portfolio variance for each of the target returns μ which is now just a scalar. It can be done using [2]:

$$\min_{w \in R^D} \; w^\top \sum w \tag{3.4}$$

subject to the following constraints:

$$w^T \boldsymbol{\mu} = \mu \tag{3.5}$$

and

$$w^T \mathbf{1} = 1 \tag{3.6}$$

Where w is a vector of portfolio weights, $\sum$ is a covariance matrix, μ is a target return and $\boldsymbol{w}$ is a vector of expected returns for each asset. The constraints come from the fact that we only take into consideration portfolios whose expected return is equal to a specific μ, and the second constraint is redefined from before, simply forcing all the portfolio weights to sum up to one. Now, it would be impractical to solve such an optimization problem by hand, given that there are 200,000 simulated portfolios. That is why it was done using Python code, which can be seen in Listing 3.4. First of all, an array of 100 target returns between a minimum and maximum was created, meaning that the final efficient frontier will consist of 100 points. The objective functions are then defined in lines 6, 9, and 12, corresponding to equations (2.4), (2.5), and (2.6), respectively. A constraints dictionary is then defined for the optimizer, enforcing the functions defined above. The optimizer is then initiated for the first target return in line 28, taking in all the functions and constraints defined above, as well as the bounds shown in Listing 3.3. The outputs are the optimal portfolio weights (`res.x`) and the minimum portfolio variance (`res.fun`) for that specific target return μ. The code then runs through all 100 target returns by updating the constraint (2.5) for every iteration, finding the portfolio weights that give the minimum variance for that target return. These minimum variance values for all μ are stored in the optimized risks list, line 38. Finally, the square root of these variances is taken to obtain the actual portfolio risk (standard deviation).

An efficient frontier usually has a shape of a hyperbola, as in a (μ, σ^2) plane, variance is a quadratic function of the mean, resulting in a parabola, however since risk (σ) is a parameter of interest, it is transformed into a (σ, μ) plane, where σ is a function of a square root of μ, which is a hyperbola [3]. A typical Efficient Frontier can be seen in figure 3.2.

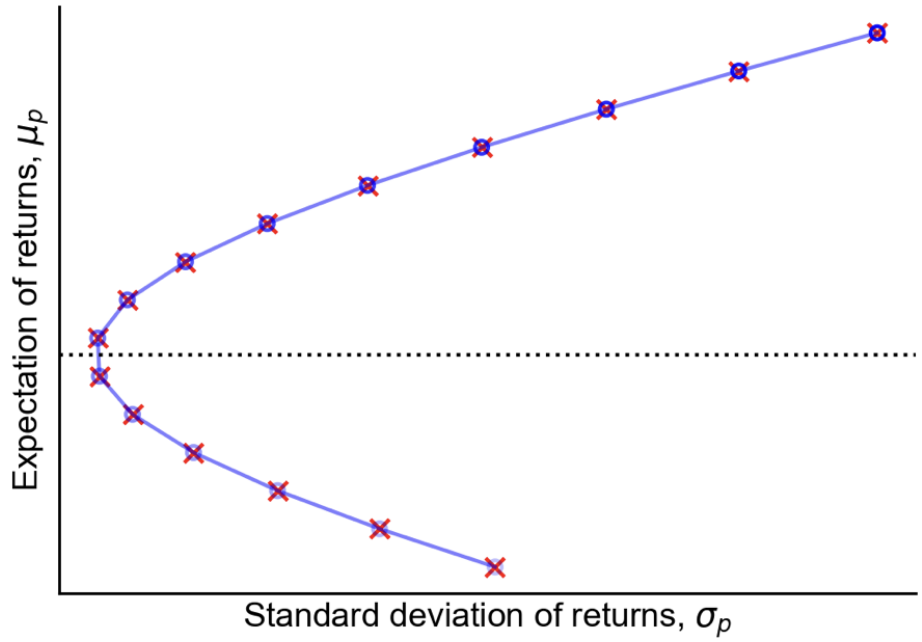

Figure 3.2: This Shows a Usual Hyperbolic Shape of an Efficient Frontier, Also Known as a Markowitz Bullet [3].

```
#Plotting an efficient frontier

target_returns = np.linspace(min_return, max_return, 100)
from scipy.optimize import minimize

def get_portfolio_variance(weights):
    return weights.dot(cov.dot(weights))

def target_return_constraint(weights, target):
    return weights.dot(mean_return) - target

def portfolio_constraint(weights):
    return weights.sum() - 1

#Defining a constraint dictionary

constraints = [
    {
     'type': 'eq',
     'fun': target_return_constraint,
     'args': [target_returns[0]],
     },
    {'type': 'eq',
     'fun': portfolio_constraint
     }
```

```
    ]

res = minimize(
    fun = get_portfolio_variance,
    x0 = np.ones(D)/D,
    method = 'SLSQP',
    constraints = constraints,
    bounds = bounds,
    )

#Running through all the target returns, to generate an
   efficient frontier

optimized_risks = []
for target in target_returns:
    constraints[0]['args'] = [target] #Each iteration
       enforces a different required return
    #Pasting a minimizing function from before
    res = minimize(
        fun = get_portfolio_variance,
        x0 = np.ones(D)/D,
        method = 'SLSQP',
        constraints = constraints,
        bounds = bounds,
    )
    if res.success:
        optimized_risks.append(np.sqrt(res.fun))
```

Listing 3.4: Efficient Frontier Plot

Sharpe Ratio

Sharpe Ratio is a measure of an investment's performance adjusted for risk. It compares the realized return with the return possible with a risk-free rate over the same time period [4].

$$SharpeRatio = \frac{R_p - R_f}{\sigma_p} \tag{3.7}$$

Where R_p, R_f and σ_p are the portfolio return, return of an investment under the risk free rate and the standard deviation of an excess portfolio return, respectively [4]. Generally, higher Sharpe Ratios mean that the risk-adjusted return is more attractive. This means that one of the portfolios on the efficient frontier will have the most attractive Sharpe Ratio, hence the final part of the code is aimed at determining it.
First of all, a risk-free interest rate needs to be chosen. It has been decided that a 10-year Dutch Government Bond Yield will be chosen as a suitable risk free rate, which as of October 17th 2025 is 2.7394% [5].

An objective can be summarized as the following [6]:

$$\max_{\mathbf{w}} \quad \frac{\mathbf{w}^T \boldsymbol{\mu} - R_f}{\sqrt{\mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w}}} \tag{3.8}$$

It is written as a function in listing 3.5 in line 4. It is subject to the constraint (2.6) and the bounds in listing 3.3. An optimizer function in Python scipy library can only minimize an objective function, while we need to maximize it. In order to resolve that, one can simply minimize the negative Sharpe Ratio, which is the same as maximizing. This is exactly what the code in listing 3.5 does in line 10. Finally the best Sharpe Ratio is printed as -res.fun (as it is negative), and the corresponding portfolio weights res.x. Optimal risk and optimal return are then calculated in lines 24 and 25, in order to be able to plot that point in a graph.

```
risk_free_rate = 0.027394

#for maximization
def neg_sharpe_ratio(weights):
    mean = weights.dot(mean_return)
    sd = np.sqrt(weights.dot(cov).dot(weights))
    return -(mean-risk_free_rate)/sd

#Minimize function (to optimize SR, since it is negative)
res = minimize(
    fun = neg_sharpe_ratio,
    x0 = np.ones(D)/D,
    method = 'SLSQP',
    constraints = {
        'type': 'eq',
        'fun': portfolio_constraint
        },
    bounds = bounds
    )

best_sr, best_w = -res.fun, res.x


opt_risk = np.sqrt(best_w.dot(cov).dot(best_w))
opt_ret = mean_return.dot(best_w)
```

Listing 3.5: Maximum Sharpe Ratio

Chapter 4

Results

4.0.1 Results - Generic Case, One-Year Time frame

As all the needed values are calculated, they can be plotted in a graph. Figure 4.1 shows the simulated portfolios, efficient frontier and the maximum Sharpe ratio for a one-year time frame from 09-09-2024 to 08-09-2025. Short Selling was allowed for this case.

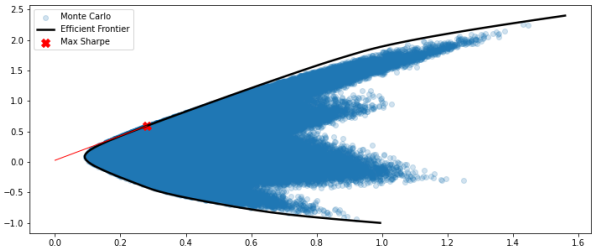

Figure 4.1: Mean-Variance Optimization with Maximum Sharpe Ratio, Short Selling Allowed

The corresponding asset allocations can be seen in table 4.1.

Asset	**Value**
Meta	0.4442
BTC	0.4780
ASML	-0.2233
ETF	-0.0268
Toyota	0.0609
Bond	0.2669

Table 4.1: Asset Values Summary, Generic Case

4.0.2 Results - Bear Market

It will now be checked whether the algorithm allocates higher weights to 'safe' assets during an actual bear market. From the code, the most significant bear market takes place from 19-02-2025 until 08-04-2025. These dates can be substituted into the code, leading to figure 4.2.

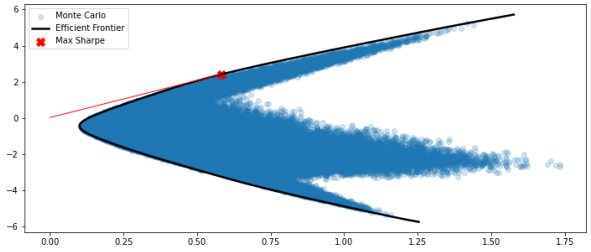

Figure 4.2: Mean-Variance Optimization With Maximum Sharpe Ratio, Bear Market, Short Selling Allowed

With corresponding weights being:

Asset	**Value**
Meta	-0.5000
BTC	0.1399
ASML	-0.5000
ETF	-0.5000
Toyota	0.9082
Bond	1.4519

Table 4.2: Asset Values for Bear Market 1 Summary, Short Selling Allowed

These results make perfect sense and confirm the hypothesis stated at the beginning. The algorithm allocated the greatest weight to a Government Bond. The weight for Toyota is also relatively high, indicating that this particular stock was not affected by the bear market. The rest of the assets have either the greatest possible negative weight (-0.5), meaning they were shorted, or a very small positive weight in the case of Bitcoin. The maximum Sharpe Ratio for this case was 4.096, which is considered exceptional, meaning that the investment's return was around four times its volatility.
The results for the same time frame where the short selling wasn't allowed is presented in figure 4.3

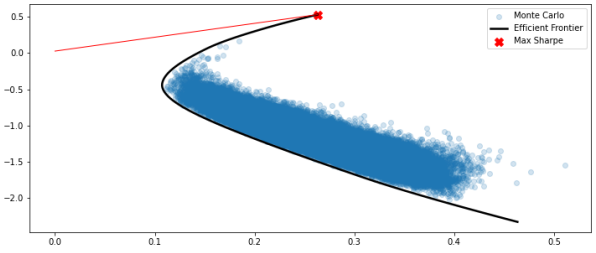

Figure 4.3: Mean-Variance Optimization with Maximum Sharpe Ratio, Bear Market, Long Positions Only

With corresponding weights being:

Asset	Value
Meta	0.0000
BTC	0.0000
ASML	0.0000
ETF	0.0000
Toyota	0.0000
Bond	1.0000

Table 4.3: Asset Values for bear market 1 Summary, Long Positions Only

Once again, the results make perfect sense, despite being quite extreme. The entire investment was allocated to the Government Bond, which received a weight of 1. All other assets ended up with a weight of 0. The maximum Sharpe Ratio for this case was 1.919, meaning that the investment's return was almost twice its volatility, which is still considered very good when the market is turbulent.

Finally, a case of a different bear market must be checked to confirm that the algorithm functions consistently. A relatively short bearish period took place from 2024-12-09 to 2025-01-13, which can also be seen in Figure 3.1. Plotting an efficient frontier:

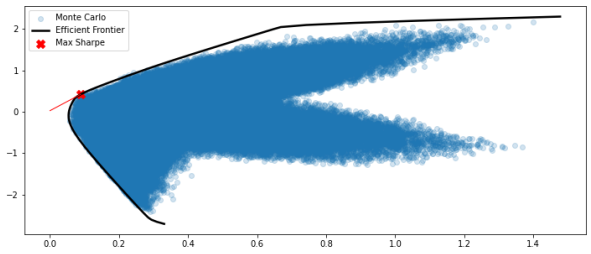

Figure 4.4: Mean-Variance Optimization with maximum Sharpe Ratio, Bear Market 2, Short Selling Allowed

With the following asset allocations:

Asset	Value
Meta	0.0578
BTC	0.0065
ASML	0.3718
ETF	-0.5000
Toyota	0.1227
Bond	0.9413

Table 4.4: Asset Values for Bear market 2 Summary, Short Selling Allowed

Once again, the algorithm allocated the greatest weight to the Government Bond, while the stocks, ETF, and cryptocurrency were either given very small weights or were shorted. The only exception this time was ASML stock, which had a weight of 0.3718, meaning it was not affected by that specific bear market. The maximum Sharpe Ratio was 4.428,

which can also be considered exceptional, meaning that the investment's return was almost 4.5 times its volatility.

Chapter 5

Conclusion

Concluding this report, the aim was to determine how portfolio weights of different assets differ between market regimes, and what their optimal allocation is. It can therefore be concluded that the Hidden Markov Model performs well in identifying market regimes, while the mean–variance portfolio optimization algorithm efficiently allocates assets once a regime has been identified. When the market is unusually volatile and the mean returns are negative, it allocates greater weights to safer assets such as U.S. Government Bonds, and shorts the majority of stocks when short selling is allowed by the user. On the other hand, when the market trend is generally increasing and volatility is low, the algorithm allocates greater weights to assets with higher returns, such as stocks and cryptocurrencies. This therefore confirms the hypothesis and indicates that the algorithm has achieved its purpose.

The potential implications of this algorithm may be in the field of risk management, as it could be used for early detection of market regimes and dynamic adjustment of the portfolio—allowing increased exposure to stocks and cryptocurrencies during bull markets, and greater exposure to bonds during bear markets.

Finally, this project can be improved by taking more assets and asset classes into consideration, potentially adding more stocks and including commodities such as gold as a safe asset, as well as options. Another improvement could be to make the model compute the initial estimates for mean, variance, and transition probabilities based on the data instead of defining them manually. Despite the chosen parameters being fairly accurate with respect to the general market, fitting the model to the data to estimate them would improve accuracy even further.

Bibliography

[1] MarketWatch, *Financial News, Stock Market Data, and Analysis*, Available at: https://www.marketwatch.com, accessed October 2025.

[2] Emilio Barucci, *Financial Markets Theory*, Springer Finance, 2017.

[3] Gundersen, G. (2022). *Understanding the Markowitz Efficient Frontier*. Retrieved from https://gregorygundersen.com/blog/2022/03/14/markowitz/

[4] Investopedia, *Sharpe Ratio: Definition, Formula, and Examples*, Available at: https://www.investopedia.com/terms/s/sharperatio.asp, accessed October 2025.

[5] Trading Economics, *Risk-Free Interest Rate NL*, Available at: https://tradingeconomics.com/netherlands/government-bond-yield, accessed October 2025.

[6] Bookdown, *7.2 Maximum Sharpe Ratio Portfolio*, Available at: https://bookdown.org/palomar/portfoliooptimizationbook/7.2-MSRP.html, accessed October 2025.

Appendix A

Appendix

The entire project code can be found in the Github: GitHub Repository