Portfolio Creation and Optimization through Machine Learning

Hephaestus Applied Artificial Intelligence Association

Bocconi Students Investment Club

June 2024

Abstract

In this study, we propose a two-step methodology for the creation and optimization of investment portfolios utilizing machine learning techniques. Our analysis focuses on S&P 500 stocks from 2008 to 2022, incorporating fundamental data, stock prices, and macroeconomic variables as features in our models. The initial phase involves feature selection through Principal Feature Analysis (PFA), a dimensionality reduction algorithm. These selected features are subsequently used to predict stock price movements (upward or downward) via the XGBoost classification algorithm. The subsequent phase entails selecting stocks based on their forecasted return, and constructing optimal portfolios through both mean-variance and mean-entropy optimization approaches.

Head of Project: Filippo A. Ronzino, *Members*: Francesco Braicovich, Lorenzo Calda, Ettore Conti, Andrea Franceschini, Giuseppe Iannone, Benjamin Jilma, Francesco Sassi, Teodor Tankov

Contents

1. Introduction

1.1. Motivations

In today's rapidly evolving financial landscape, the importance of managing investments effectively cannot be overstated. The ability to make informed investment decisions is paramount for investors seeking to navigate the complexities of the stock market. The problem with this is that investing is not solely about maximizing returns, but also, and perhaps mainly, about mitigating risks. Many mathematical models were created and developed in the past decades to find the optimal bundle composed of the most returns relative to a reasonable amount of risk; this scientific approach to finance calls for an evolution: the implementation of Artificial Intelligence in the decision-making process of structuring and managing a portfolio.

In this study, we aim to design a model that analyzes companies included in the S&P 500 since 2008, utilizing macroeconomic, fundamental, and price data. The project is divided into two fundamental parts: portfolio construction and optimization.

For the portfolio construction component, we select the top 20 stocks predicted to perform best during the holding period. Although transaction fees are not explicitly accounted for in our model, turnover constraints are implemented to limit the transaction costs associated with portfolio rebalancing.

To forecast future stock prices, we constructed a dataset comprising approximately 100 features. Dimensionality reduction was performed using Principal Feature Analysis (PFA), an algorithm combining Principal Component Analysis (PCA) and K-Means clustering [3]. An XGBoost classifier was then trained on these features to predict the return sign of stocks for the future holding period. The training and holding periods for the model are set to 2 years and 3 months, respectively, based on the assumption that regime shifts in financial markets occur approximately every 2 years, necessitating model retraining.

In the portfolio optimization phase, we compared the classical Markowitz Mean-Variance Optimization [4] with a more recent model, the Mean-Entropy approach [5].

1.2. Dataset

For our dataset, corporate data for companies listed in the S&P 500 were sourced from the Wharton Research Data Service [6]. The dataset underwent a cleaning process, with stock prices adjusted for splits and dividends. In addition to corporate data, our dataset includes the 1-month Federal Funds Futures as a macroeconomic variable, as well as price data including volatility, price, and volume. Subsequently, we generated the final features, which included momentum signals and price-based factors such as the Moving Average Convergence Divergence (MACD) and the Relative Strength Index (RSI).

The RSI was computed using the formula:

$$\text{RSI} = 100 - \left[\frac{100}{1 + \frac{\textit{Avg. gain}}{\textit{Avg. loss}}} \right]$$

and the MACD was calculated as follows:

$$\mathrm{MACD} = \mathrm{EMA}_{12} - \mathrm{EMA}_{26}$$

where EMA represents the Exponential Moving Average, which places greater weight on more recent data points. The EMA is defined by the equation:

$$\mathrm{EMA}_t = \mathrm{Price}_t \cdot \frac{\mathit{Smoothing}}{1 + \mathit{Periods}} + \mathrm{EMA}_{t-1} \times \left(1 + \frac{\mathit{Smoothing}}{1 + \mathit{Periods}}\right)$$

We applied logarithmic transformations to features such as volume, to normalize the data across different orders of magnitude for each stock and various trading days, thereby minimizing interference in future price forecasts.

A critical aspect of our dataset creation was the stationarization of raw data. Stationarity is essential in time series analysis to ensure consistent variance over time, enabling the model to accurately identify relationships within the data without bias, thus enhancing forecast accuracy.

To achieve stationarity, we employed fractional differentiation. This technique maintains the maximum memory of the time series while achieving stationarity. A detailed discussion on the advantages of fractional differentiation over integer differentiation is provided in [2].

2. Predictive Model

This section elucidates the process of feature selection and the workings of the predictive model. The primary aim of feature selection is to reduce the dataset's size, thereby enhancing processing speed and manageability without losing critical information or compromising the model's performance. In our approach, we utilize two years of data to identify the most independent features, which are then employed to train the model to predict the sign of the stock return over the subsequent n days (the holding period). The model trained on this selected set of features is used for two years before a new model is trained and applied to the latest data.

2.1. Principal Feature Analysis

To perform feature selection, we resort to Principal Feature Analysis [3], which leverages Principal Components Analysis (PCA) and K-Means, as explained below. PFA is a technique used to identify a subset of features from a high-dimensional dataset that preserves most of the variability present in the original dataset. Given a zero-mean n-dimensional random feature vector X with covariance matrix Σ, PFA follows these steps:

1. **Principal Components:** Compute the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and eigenvectors of Σ. Arrange the eigenvalues in descending order $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. Let A be the matrix of orthonormal eigenvectors of Σ such that:

$$\Sigma = A\Lambda A^T, \quad \Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

2. **Subspace Dimension:** Choose a subspace dimension q to retain a desired level of variability in the data. Construct the matrix A_q using the first q columns of A. The retained variability is given by:

$$\text{Variability Retained} = \frac{\sum_{i=1}^{q} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \times 100\%$$

3. **Feature Clustering:** Project the original features into the lower-dimensional subspace to obtain vectors $V_i \in \mathbb{R}^q$, which represent the projections of the i-th feature. Cluster the absolute value of these vectors $\{|V_1|, |V_2|, \ldots, |V_n|\}$ into $p \geq q$ clusters using the K-Means algorithm with Euclidean distance.

4. **Principal Features:** For each cluster, select the feature x_i corresponding to the vector V_i closest to the cluster mean as a principal feature. This ensures that the selected features are representative of the variability in the lower-dimensional space while minimizing redundancy.

We use PFA to find the best features to train XGBoost every two years. We select the features feedinf the last two years of data to the PFA algorithm described above.

2.2. XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm based on the gradient boosting framework. It enhances the predictive accuracy of a series of weak classifiers (typically decision trees) by combining their outputs. Here, we provide a detailed technical explanation of how XGBoost operates, with a focus on its use in classification tasks and the role of regularization in improving prediction robustness.

2.2.1 Model Framework

Given a dataset $\{(x_i, y_i)\}_{i=1}^N$, where x_i represents the feature vectors and $y_i \in \{0, 1\}$ represents the binary class labels, XGBoost aims to predict $P[y_i = 1|x_1]$, i.e. the probability of y_i being 1. The model consists of an ensemble of K trees, and the prediction for the i-th instance is given by:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$$

where $f_k \in \mathcal{F}$ is the space of regression trees (also known as CART) [1].

2.2.2 Objective Function and Regularization

The objective function of XGBoost combines the loss function measuring prediction error and a regularization term controlling model complexity:

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \ell(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where ℓ is the loss function and Ω is the regularization term. For binary classification, the *logistic loss function* is used:

$$\ell(y_i, \hat{y}_i) = -\left[y_i \log(\sigma(\hat{y}_i)) + (1 - y_i) \log(1 - \sigma(\hat{y}_i))\right]$$

where $\sigma(\hat{y}_i) = \frac{1}{1+e^{-\hat{y}_i}}$ is the sigmoid function.

Regularization in XGBoost consists of two parts: L1 regularization (Lasso) and L2 regularization (Ridge), which control the complexity of the trees:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where T is the number of leaves in the tree, w_j are the leaf weights, γ is the complexity parameter controlling the number of leaves, and λ is the L2 regularization term. Including the regularisation terms in the objective function promotes simpler models which are able to generalise better. Moreover XGBoost implements column subsampling and a learning rate to further prevent overfitting [1].

2.2.3 Tree Construction

XGBoost builds trees in a greedy manner, adding one tree at a time. At each step, it minimizes the following regularized objective, using a second-order Taylor approximation of the loss function we will not discuss:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{N} \ell(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

To find the optimal structure of the tree, XGBoost evaluates the gain for each split and chooses the one that maximizes the gain, which can be derived to be:

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

where G_L and G_R are the sums of the first-order gradients and H_L and H_R are the sums of the second-order gradients for the left and right splits, respectively. The optimal leaf weights are updated as:

$$w_j^* = -\frac{G_j}{H_j+\lambda}$$

2.2.4 Implementation, Fine-Tuning & Stock Selection

Every two years we train a new XGBoost model: feeding it 1 years and 9 months of stock market data, spanning from January 1st to September 30th (not 31st December to avoid bias). We train the model to classify the direction (sign) of the future 90-day return of a stock, utilizing only the single day features selected by the Principal Feature Analysis (PFA) algorithm.

To optimize the model, we perform hyperparameter tuning with stratified k-fold cross-validation and grid search. This process involves systematically evaluating a predefined grid of hyperparameter values to identify the optimal combination that maximized the model's predictive performance while ensuring robustness. The use of stratified k-fold cross-validation helps maintain balanced class distributions across the folds, providing a reliable estimate of the model's generalization capability.

The same XGBoost model is used to predict stock returns for two years until a new model is trained. Given a holding period n (which we tested at 90 and 45 days), the trained model is used to predict the probability of positive return every n days. All the stocks that have probability of positive return higher than the 0.999 threshold are selected to trade.

3. Portfolio Selection and Optimization

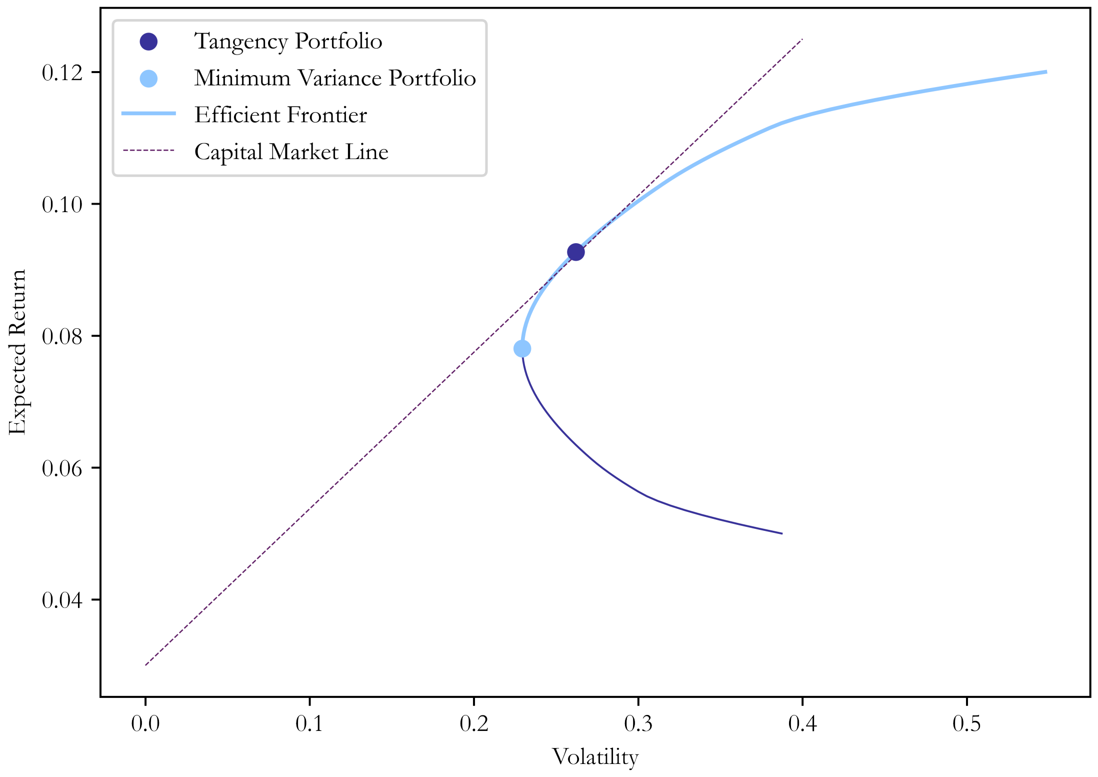
This section will be devoted to explaining how the portfolios, based on the prediction of our model, are constructed. We seek to compare the output of the traditional mean-variance approach (MVPO) to those obtained by implementing a more recent model as explained by [5], the return-entropy portfolio optimization (REPO). The reason for this comparison is to show how a different definition of risk (variance or entropy) can lead to different investment decisions.

3.1. Mean-Variance Portfolios

Here we present the classic mean-variance approach to portfolio optimization that stems directly from the work of Harry Markowitz in *Portfolio Selection* [4] and upon which an important branch of financial literature is based.

The intuition underlying this model is that investors like expected returns and dislike variance, where variance is the model's measure of *risk*. This definition of risk is economically consistent with a quadratic loss function in forecasting models where, for example the mean squared forecast error of the time-h forecast is defined as: $\text{MSFE}\left[\hat{y}_t(h)\right] \equiv \mathbb{E}\left[\left(y_{t+h} - \hat{y}_t(h)\right)^2\right]$.
In this framework, portfolios are selected on the *efficient frontier* as shown in fig. 3.1.

Figure 3.1. Example of Efficient Frontier



Sources: BSIC, Hephaestus

The efficient frontier is constructed by finding, among the possible combinations of assets, those who minimize variance for every level of expected return, or equivalently maximize expected return for every level of variance. As seen in fig. 3.1, the resulting efficient frontier will be the upper arm of the frontier formed by the feasible portfolios. Now, by defining μ as a vector of expected returns, w as a vector of weights and Σ as the variance-covariance matrix of the assets, the efficient frontier can be expressed as follows by iterating for different levels of p, the expected returns of the portfolios:

$$\begin{aligned} \min_{w} \quad & w'\Sigma w \\ \text{s.t.} \quad & w'\mu = p \\ & w'\mathbf{1} = 1 \end{aligned} \tag{3.1}$$

In particular, at each point in time, we compute, among the best stocks according to the model presented in section 2 the combination of weights that maximize the Sharpe Ratio and the one that minimizes the portfolio variance. The first portfolio is the *tangency portfolio* in fig. 3.1; by defining the Sharpe Ratio as $SR = \frac{\mathbb{E}[r_p - r_f]}{\sigma_p}$, where r_p, r_f, σ_p are respectively the portfolio's return, risk-free rate and the portfolio volatility, our optimization becomes:

$$\begin{aligned} \max_{w} \quad & \frac{w'\mathbb{E}[r - r_f]}{\sqrt{w'\Sigma w}} \\ \text{s.t.} \quad & w'\mathbf{1} = 1 \end{aligned} \tag{3.2}$$

Where $\mathbb{E}[r - r_f]$ is just the vector of expected excess returns of the selected securities and the Sharpe Ratio represents the excess return gained per unit of risk.
The second portfolio is the *minimum variance portfolio* in fig. 3.1 and is obtained by solving:

$$\begin{aligned} \min_{w} \quad & w'\Sigma w \\ \text{s.t.} \quad & w'\mathbf{1} = 1 \end{aligned} \tag{3.3}$$

Which just finds the absolute minimum level of variance among the feasible portfolios and not for each level of return contrary to eq. (3.1). We run this optimization every time we rebalance our strategy to compare the performance of the different optimal portfolios over time.

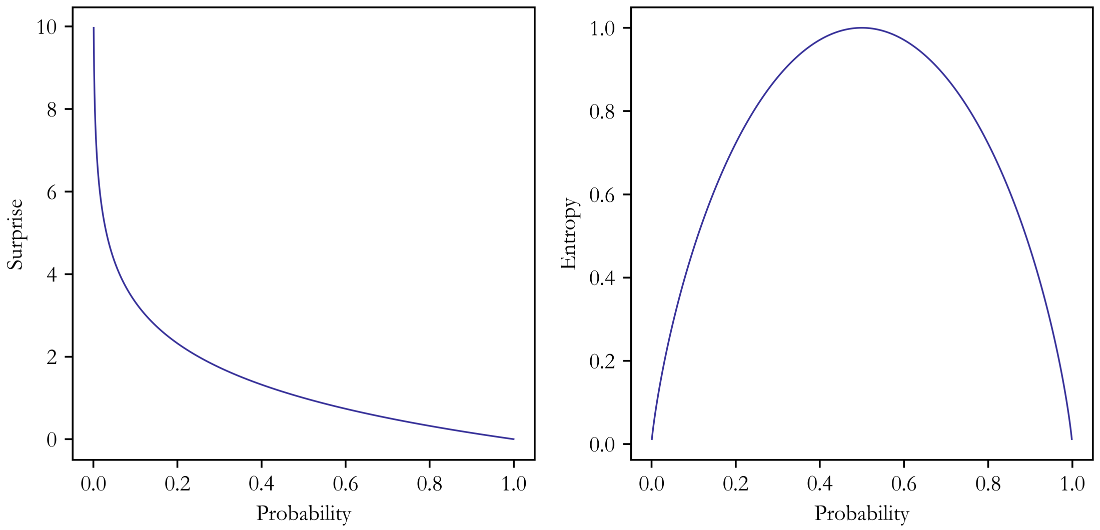
3.2. Return-Entropy Portfolios

3.2.1 What is Entropy?

Before dealing with how portfolios are constructed when changing the definition of risk, we present to the reader a brief introduction to the concept of *entropy* as presented by Shannon in his seminal paper from 1948 *A Mathematical Theory of Communication* [7]. As a concept born from information theory, entropy essentially measures the amount of randomness inherent in a probability distribution. To present entropy in an intuitive way, we introduce the rather intuitive concept of "surprise": if an event has a probability p of happening, we denote the surprise as $\log\left(\frac{1}{p}\right) = -\log(p)$, this way if an event has probability of 1, the surprise is 0, while for unlikely events, the surprise increases being not defined for impossible events. Entropy is nothing more than the expected value of surprise, or the average information produced by a random variable X, and we denote it by $H(X)$.

$$H(X) \equiv \mathbb{E}\left[-\log\left(P(X)\right)\right] = -\sum_{i=1}^{n} P(x_i)\log\left(P(x_i)\right) \tag{3.4}$$

Figure 3.2. Surprise and entropy as a function of probability for two events with probabilities p and (1-p).



Sources: BSIC, Hephaestus

As can be seen in fig. 3.2, entropy is bounded by the maximum entropy distribution where each event has the same probability (the uniform distribution), in the chart the maximum is clearly reached for $p = 0.5$. More formally, for a random variable X with m states:

$$H(X) = \mathbb{E}\left[-\log\left(P(X)\right)\right] \leq \log\left(\mathbb{E}\left[\frac{1}{P(X)}\right]\right) = \log(m) \tag{3.5}$$

Therefore, while variance is potentially ubounded as a risk measure, entropy is always bounded by the uniform distribution.

3.2.2 From Mean-Variance to Return-Entropy

As expressed in [5], mean variance optimization in practice suffers from five main drawbacks:

1. Optimal solutions assigning large weights to high-risk assets,
2. Disturbing the dependence structure equilibrium when introducing forward-looking estimates instead of the mean,
3. Variations in results when adjusting the inputs,

4. Inability to deal with non-normal and asymmetric distributions,
5. Difficulty in estimating expected returns and the variance-covariance matrix.

The introduction of entropy as a risk measure easily overcomes the issue of dealing with non-normal distributions as entropy accommodates non-symmetric distributions and is fully non parametric. Furthermore, it provides greater diversification in optimal portfolios while producing more stable solutions and maintaining the dependence structure[1], as shown by [5].

The first step to building RE optimal portfolios is building the probability distribution of the returns. To this end we build empirical distributions for each asset by partitioning the set of historical observations in m states $A_1, \dots, A_m$ where A_k is the interval $[a_{k-1}, a_k]$, and we estimate the probability of the return being in A_k:

$$\hat{f}_T(r,k) = \frac{1}{T}\sum_{j=1}^{T} \mathbb{I}_{(a_{k-1}<r_j\leq a_k)} \tag{3.6}$$

Where T is the time period and r is the vector of historical returns. For a portfolio of n assets, by defining w as the $1 \times n$ vector of weights and R as the $T \times n$ matrix of assets' returns, we obtain the vector of portfolio returns $R_p = wR'$ and the probability generating function becomes:

$$g(x, R_p) = \frac{1}{T}\sum_{j=1}^{T} x^{\{k:\, a_{k-1}<R_{p_j}\leq a_k\}} \tag{3.7}$$

Where the coefficients of x^k represent the empirical probability of $R_P \in A_k$ and are obtained from g by taking the $k-th$ derivative of g, $g^{(k)}(0)$ at $x = 0$ divided by $k!$:

$$\frac{g^{(k)}(0)}{k!} = \hat{f}_T(R_p,k) = \frac{1}{T}\sum_{j=1}^{T} \mathbb{I}_{(a_{k-1}<Rp_j\leq a_k)} \approx P\left(R_p \in A_k\right) \tag{3.8}$$

And therefore by substituting $\hat{f}_T(R_p,k)$ in eq. (3.4):

$$H(R_p) = -\sum_{k=1}^{m} \hat{f}_T(R_p,k)\log\left(\hat{f}_T(R_p,k)\right) \tag{3.9}$$

And the optimization problem becomes the multi-objective minimization of entropy and maximization of expected returns as follows:

$$\begin{aligned} \min_{w} \quad & H(R_p) = -\sum_{k=1}^{m} \frac{g^{(k)}(0)}{k!}\log\left(\frac{g^{(k)}(0)}{k!}\right) \\ \max_{w} \quad & \mathbb{E}\left[R_p\right] = w'\mu \\ \text{s.t.} \quad & w'\mathbf{1} = 1 \end{aligned} \tag{3.10}$$

Where μ is the vector containing the expected returns of the assets.

[1]Since entropy is not dependent from the mean.

Let us now look at the practical implementation [2]. The main problem with this theoretical approach is that computing the entropy of the distribution of returns of the entire weighted portofolio would be complex, since we do not know *a priori* the weights. To tackle this issue, we computed the entropy of the distribution of returns for each stock, and we then minimized the weighted sum of the entropies. After getting from the models the stock identifiers (*permno*) "to buy", the corresponding price data for each permno is processed, and we use them to compute returns distribution through KDE. In fact, we estimate the empirical probability described in [5] through a smoothed estimate of the probability distribution of returns, which captures the shape and variability of the data.

Hence, the optimization problem is formulated with the following components:

- *Weights Variable (w)*: a variable representing the portfolio weights for each stock. These weights determine the proportion of the portfolio allocated to each stock.
- *Expected Returns*: the expected returns for each stock.
- *Entropy Measures*: entropies for each stock's return distribution, computed using the KDEs. These entropies are weighted by a small λ value to balance their influence in the optimization.

Indeed, we equivalently translate the multi-objective optimization described in eq. (3.10) into maximizing the sum of weighted expected returns and weighted entropies, which is, indeed, the equivalent of maximizing expected returns while minimizing the entropy (i.e. maximizing minus the entropy). To be precise we added to our variable w another constant weight λ to the entropy in order to find a good trade off between maximizing expected return and minimize entropy, because of the different order of magnitudes between returns and entropies.

We then use the following constraints:

- *Sum of Weights*: the sum of the portfolio weights must equal 1, ensuring that the entire capital is allocated, this translates into $\sum_{i=1}^{n} w_i = 1$, or in vector notation as in (3.10), $w\mathbf{1} = 1$
- *Non-Negative Weights*: each weight must be non-negative, preventing short positions, hence for all i, $w_i \geq 0$
- *Weight Upper Bound*: each weight is constrained to be less than or equal to 0.25, promoting diversification and preventing over-concentration in any single stock. This amounts to adding as a contraint, for all i, $w_i \leq 0.25$

Then the `cvxpy` library is used to solve the convex optimization problem and the optimal portfolio weights are extracted from the solution.

[2]Small deviations from the [5] are described and used.

4. Results and Performance

4.1. Interpreting the predictions of XGBoost

In this subsection, we interpret the predictions by comparing the performance of an equally weighted portfolio of selected stocks to that of the S&P 500. Initially, the returns of our portfolio appear similar to those of the S&P 500, albeit with different beta exposure.

To understand this relationship, we regress the returns of our equally weighted portfolio against those of the S&P 500. This regression results in a poor fit, characterized by a low R^2 value and a statistically insignificant beta. The reasons for this discrepancy are discussed later in the paper.

The primary reason for the similarity in returns between our portfolio and the S&P 500 is the model's objective: to classify whether a stock will increase or decrease over a specified timeframe. Our model, trained on all stocks within the S&P 500, minimizes loss during training by classifying stocks with the highest aggregate covariance. Consequently, the model effectively classifies stocks highly correlated with the index, i.e., those with a high beta, even though the index is not equally weighted. Therefore, our strategy tends to select stocks with high covariance relative to the index, resulting in an equally weighted portfolio with a theoretically high beta.

To address the issue of poor regression fit between our strategy and the index, we must consider the frequency and manner of our model's retraining. Each retraining alters the interpretation of data by the classification trees, leading to changes in beta. This approach effectively resolves the issue.

To demonstrate this, we constructed a composite future by stitching together betas from each retrained model iteration. While this future is backward-looking and does not allow us to infer whether our strategy generates alpha by altering apparent beta, it illustrates that our strategy is strongly correlated with S&P 500 returns. This correlation explains why our strategy's absolute returns are significantly high over the backtested timeframe.

4.2. Portfolio Performances

We now turn our attention to the results our strategy yielded. The various backtests we implemented all share the same stock selection procedure but differ in terms of which portfolio optimization technique is used and which holding period is considered.

Before delving into the results of the backtests, we first focus on the metrics achieved by XGBoost over the different periods: In these tables we test the performance of the models we trained on never seen before data; meaning that the first row corresponds to the first model, which was trained on data from 2008 and 2009 and used to predict the data starting from the beginning of 2010 and ending at the end of 2011. As we can see the task of accurately predicting stock direction is not a trivial one, the trained XGBoost models achieve a maximum accuracy of only 54.5% over the different periods, just barely better than choosing randomly. However, upon further analysis, we realize this disappointing general performance is driven mostly by a lower ability of the models to predict stocks which will have negative direction; in fact, the precision and recall metric indicate that the models' ability to correctly identify stocks which will go up is far better than that of identifying stocks which will go down. Consequently, we expect that shorting using this strategy will not yield results as good as going long.

Table 4.1: XGBoost results

Testing Start Date	Accuracy	Precision	Recall
2010-01-01	0.523	0.634	0.520
2012-01-01	0.513	0.680	0.528
2014-01-01	0.531	0.557	0.718
2016-01-01	0.525	0.687	0.544
2018-01-01	0.545	0.562	0.760
2020-01-01	0.541	0.684	0.560

Source: BSIC, Hephaestus, WRDS

We now turn our attention to backtesting. We first note that in all our backtests we assumed zero transaction costs, we also note that, since our models have difficulty making shorting decisions, we only consider a long-short portfolio in the equal weights case.

Table 4.2: Strategy Results from 2010 to 2022

Benchmark			Cumulative Return	Yearly Return	Standard Deviation	Sharpe Ratio
S&P 500			428.38%	14.88%	0.16	0.73
Portfolio Weights	**Trading Type**	**Holding Period (days)**	**Cumulative Return**	**Yearly Return**	**Standard Deviation**	**Sharpe Ratio**
Return-Entropy	Long only	90	389.39%	14.15%	0.25	0.58
Equal Weights	Long only	90	684.71%	18.73%	0.19	0.90
Equal Weights	Long-Short	90	155.90%	8.14%	0.091	0.69
Mean-Variance	Long only	90	492.25%	15.98%	0.15	0.96
Return-Entropy	Long only	45	1616.01%	26.73%	0.30	0.87
Equal Weights	Long only	45	1320.71%	24.75%	0.22	1.03
Equal Weights	Long-Short	45	273.89%	11.62%	0.13	0.77
Mean-Variance	Long only	45	604.19%	17.66%	0.16	1.01

Source: BSIC, Hephaestus, WRDS, Yahoo Finance

When calculating the Sharpe ratios we utilize the 10-Year Treasury yield as proxy for the risk free rate of return.

As mentioned above we can clearly see that long-short strategies greatly underperform both the long only strategies and the benchmark, reinforcing the idea that the models aren't able to reliably produce satisfactory shorting results. The 90-day holding period had a cumulative returns of 155.90% and a Sharpe Ratio of 0.69, indicating lower returns despite reduced volatility while the 45-day holding period improved performance, with a cumulative return of 273.89% and a Sharpe Ratio of 0.77, despite this it still lagged behind even the benchmark in terms of returns.

This trend of obtaining better performance when reducing the holding period seems to be general as each portfolio optimization strategy tested witnesses not only an increase in the return but also in the Sharpe ratio when shortening the holding period. This reinforces the common sense hypothesis that predicting shorter term stock direction is simpler than predicting longer term stock direction.

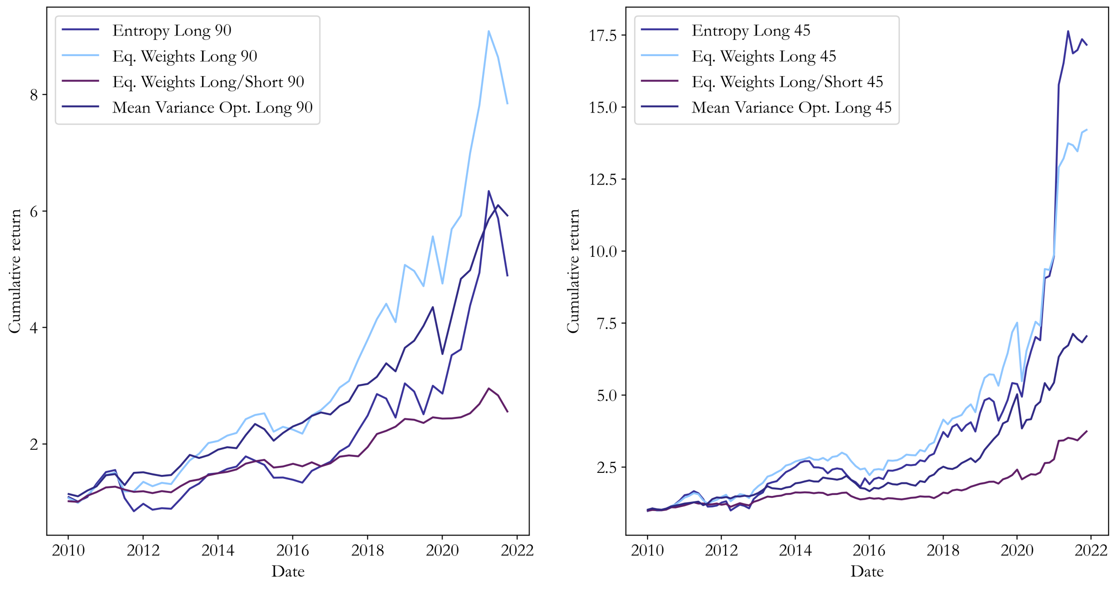
The Return-Entropy strategy showed varied results depending on the holding period. For the 90-day holding period, it yielded a cumulative return of 389.39% and a Sharpe Ratio of 0.58, underperforming the benchmark. However, with a 45-day holding period, the strategy achieved an impressive cumulative return of 1616.01% and a Sharpe Ratio of 0.87, indicating that more frequent rebalancing significantly enhanced performance despite increased risk.

The Equal Weights (Long Only) strategy consistently outperformed the benchmark in both holding periods. With a 90-day holding period, it achieved a cumulative return of 684.71% and a Sharpe Ratio of 0.90, showing strong returns and good risk-adjusted performance. When the holding period was reduced to 45 days, the strategy further excelled, reaching a cumulative return of 1320.71% and the highest Sharpe Ratio of 1.03 among all strategies, highlighting its effectiveness in maximizing returns with strong risk management.

The Mean-Variance strategy also demonstrated solid performance, especially with the 90-day holding period, achieving a cumulative return of 492.25% and a Sharpe Ratio of 0.96, outperforming the benchmark. With a 45-day holding period, the strategy produced a cumulative return of 604.19% and a Sharpe Ratio of 1.01, indicating strong returns and good risk-adjusted performance, though not as high as the best-performing Equal Weights strategies.

Below we present a graph of the cumulative return over time of the various backtests:

Figure 4.3. Cumulative return of strategies with holding period of 90 days (left) and 45 days (right)



Sources: BSIC, Hephaestus, WRDS

4.3. Further Research Questions

Further research in this field could focus on relaxing the semplifications made in the backtest, namely introducing transaction costs in the backtest.

Given the great boost in performance obtained by reducing the holding period, it could also focus on experimenting with even shorter holding periods as they appear easier to predict.

Finally with the recent advances in the field of deep learning, one could test whether building a neural network-based stock direction classifier could further increase accuracy, specifically when relating to predicting negative direction as this would allow for greater performance when shorting stocks.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016. URL `http://arxiv.org/abs/1603.02754`.

[2] Marcos López de Prado. *Advances in Financial Machine Learning*. John Wiley & Sons Inc, 2018.

[3] Yijuan Lu, Ira Cohen, Xiang Zhou, and Qi Tian. Feature selection using principal feature analysis. pages 301–304, 09 2007. doi: 10.1145/1291233.1291297.

[4] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[5] Peter Joseph Mercurio, Yuehua Wu, and Hong Xie. An entropy-based approach to portfolio optimization. *Entropy*, 22(332), 2020.

[6] Wharton Research Data Services. Corporate and non-corporate data. *WRDS website*, 2024.

[7] Claude E Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27 (3):379–423, 1948.