UCLouvain
FUCaM MONS

LOUVAIN
School of Management

Louvain School of Management

Stock Selection and Portfolio Optimization : A Machine Learning Approach

Author : Nathan SIMONIS
Supervisor : Frédéric VRINS
Academic year 2020-2021
Master in Business Engineering

Université Catholique de Louvain
Louvain School of Management

Stock Selection and Portfolio Optimization: A Machine Learning Approach

Nathan SIMONIS

Master in Business Engineering
Specialization area in Financial Engineering

LOUVAIN
School of Management

August, 2021

Abstract

A challenging problem in active asset management is the selection of stocks and their weighting to form a balanced portfolio. In this regard, we employ machine learning techniques that aim to predict which stocks will outperform their benchmark during the coming quarter. We then integrate these picks into a portfolio in such a way as to minimize its variance. Our approach is evaluated using a rolling window that allows us to measure the performance over an out-of-sample period from August 2004 to May 2021. Our empirical findings show that machine learning-based selection models can be used to construct portfolios that exhibit a higher return and a lower risk than the benchmark. The best performing model obtained a Sharpe ratio of 0.811 while the benchmark index had one of 0.681.

Keywords: Machine Learning, Stock Market, Logistic Regression, Support Vector Machine, Neural Network, Random Forest, Gradient Boosting Machine, Principal Component Analysis, Portfolio Optimization, Mean-Variance, Shrinkage, Random Matrix Theory.

Acknowledgements

I would like to express my gratitude to Prof. Frédéric Vrins for his patience, advice and for providing so many valuable opinions that contributed to the realization of this thesis.

I would also like to thank my parents as well as my close friends who reviewed and helped proofread my thesis. Moreover, they have supported me throughout the years. I am forever grateful to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The global asset management industry is a huge economic sector that was worth $103 trillion in 2020 (Lubasha et al., 2021). This industry, like many others, is going through a technological revolution with the proliferation of available data (Mike et al., 2021). As a result, while fund managers currently have access to an enormous amount of information that allows them to make more conscious decisions, the sheer quantity is such that they must now develop tools that allow them to efficiently discover patterns in this data. In this regard, machine learning, having already demonstrated its prowess in other disciplines (Zhai et al., 2021, Jiang et al., 2019), is a natural domain to consider in efforts to gain insight from the data and ultimately make predictions from it.

Fund managers generally have two major procedures to follow to perform their work successfully. Firstly, they have to select financial assets from a predefined universe. While the level of flexibility of the selection will depend on the manager's mandate, it is rather common to consider that in terms of performance, the manager should strive to outperform a pre-defined benchmark. The classical method in this regard is that a fund manager selects assets based on the financial quality and prospects of the assets' future performance. Fundamental analysis of a security aims to assess the intrinsic value of a company and then compare it to the current market price. The asset manager then judges whether this difference is sufficient to warrant investing in the security. In addition to this, technical analysis seeks to inform the manager of potential patterns in historical price and volume data that repeat themselves and could allow for predictions of future returns on financial assets. Once the selections are made, the second procedure that the manager must perform is to include them in a portfolio. The latter must be constructed

in such a way that it fulfills the manager's mandate. It is therefore subject to certain constraints, such as the prohibition of short selling, and limitations on risk. To this end, by considering that risk is quantified by the variance of returns, Markowitz (1952) proposed a framework to allocate wealth such that the variance of a portfolio is minimized for a specified minimum level of return, subject to the aforementioned constraints and potentially more.

However, these classical stock selection and portfolio allocation methods are not without criticism. The Efficient Market Hypothesis (EMH) introduced by Fama (1965) states that a market where there are a large number of participants who are rational, seek to maximize their profits, compete with one another, try to predict future asset prices, and have access to more or less the same information is deemed to be efficient. This implies that at any given time, the current price already reflects the events that have taken place as well as the ones that the market expects will take place in the future . Such a stipulation therefore refutes the technical and fundamental analysts by stating that their efforts are in vain since any mispricing would have already been accounted for. As such, an analyst would not be able to compose a portfolio of assets that would consistently outperform a portfolio composed of randomly selected stocks with the same risk level as the ones selected by the analyst. It would then be also be an exercise in futility to try to find buying and selling rules from technical indicators applicable to a stock that would result in a better return than simply buying and holding this same stock. Lo (2005) argues, however, that the EMH is not an uncompromising condition of markets but rather a characteristic that has varying degrees over time. Therefore, the author advocates an Adaptive Markets Hypothesis (AMH) in which during certain periods markets behave as if they were efficient and during others they operate in ways that are anomalous to the behavior of an efficient market. Hence, there would exist mispricings and arbitrage opportunities that only come up from time to time and last for a limited duration. Lim and Brooks (2011) therefore advocate the adoption of rolling windows when analyzing markets, thereby capturing medium-term inefficiencies and re-evaluating these periodically so as to always take into account recent market trends. These mispricings, opportunities and trends can be difficult to identify and or understand by humans given the potential non-linearity of financial markets. A substantial body of academic research is devoted to the use of machine learning techniques to predict future returns on financial assets. In

this regard, Choi et al. (2020) use 12 factors to predict future returns of 33 international markets. Once these predictions are made, they construct a portfolio where they buy the top decile of stocks with the highest expected return and sell short the bottom decile. Using a neural network, they obtain an annual Sharpe ratio approaching 2 for most of the markets considered. Paranjape-Voditel and Deshpande (2013) build a recommender system based on association rule mining. Compared to other recommendation systems, this one stands out by the fact that the recommendations are not made independently by stock but are made in the form of a portfolio where the correlation of the recommended assets are taken into account. They measure the performance of the system against the top 5 mutual funds in India and outperform them all. De Prado (2018) proposes the use of machine models as a layer of risk management applied to an existing trading strategy. Then, for given trading opportunity dates and directions, the machine learning model evaluates the current market conditions and estimates the probability that the opportunity will result in a profit. By applying this approach to a trend following strategy, Man and Chan (2020) go from a Sharpe ratio of 0.36 on the original strategy to 0.74 after the layer of risk management is applied.

The optimization framework developped by Markowitz (1952) has also been widely criticized. In part because in practice, the solutions constructed by this approach are concentrated and unstable (DeMiguel et al., 2009; Michaud, 1989; Jobson and Korkie, 1980). This is explained by the fact that the inputs of the algorithm, namely the covariance matrix and the expected return vector, are very difficult to estimate and therefore contain lots of error. Moreover, since the covariance matrix must be inverted in the optimization process, when it has a large condition number, the inversion produces results that change considerably when the input is only slightly different. Numerous researchers have investigated the issue and have proposed some potential remedies. Ma and Jagannathan (2003) show that the use of constraints that restrict the weights to be non-negative is a way to regularize the large elements of the covariance matrix. This approach helps to reduce the estimation error and enables the construction of portfolios that are just as robust as other more complex regularization techniques. Among others, Black and Litterman (1992) and Herold and Maurer (2006) propose the use of a Bayesian framework that considers the input data to the optimization procedure as unknown quantities; then the manager can utilize prior information based on expert knowledge and empirical analysis to infer the

distribution of the unknown input quantities. This approach allows one to take into account the estimation error in the construction of the portfolio; thus providing portfolios that are more readily interpretable. Other approaches call for transformations applied to the covariance matrix that aims to reduce the error before using it in the optimization procedure. Ledoit and Wolf (2004) present a new formula for estimating the covariance matrix which aims at shrinking it towards one with very little estimation error. Random matrix theory (Potters et al., 2005) considers the distribution of the eigenvalues of a random matrix. As such, by observing the eigenvalues of the covariance matrix it is possible to discern which ones are attributed to noise and which ones are attributed to signal; thus allowing the covariance matrix to be filtered from the noise (Laloux et al., 2000). Finally, De Prado (2016) considers the source of instability of the covariance matrix and shows that the more it includes assets that are interrelated, the more unstable it is. Hence, the author proposes to add a step to the optimization process that aims at clustering the highly correlated stocks. The optimization procedure is then carried out both within each group, and between each group; allowing to reduce the level of error caused by the instability.

The objective of this thesis is to explore the viability of machine learning models in both aspects of portfolio management; specifically, the selection of individual securities that are predicted to outperform a benchmark and the combination of these selections through portfolio optimization. The document is organized as follows:

- Chapter 2 presents various machine learning techniques. First, we present supervised learning algorithms that aim at learning a function that allows us to make predictions on new, unseen data. Then, we present two unsupervised learning algorithms which seek for the first to reduce the dimensionality of the data and for the second to partition it into groups of correlated variables;

- Chapter 3 showcases several methods for portfolio optimization. Starting with a theoretical overview of the classical Markowitz optimization, we describe the concerns in more detail and present 3 alternative techniques that aim at improving the solutions proposed by the optimization framework;

- Chapter 4 performs an empirical analysis of the performance of these techniques applied on a dataset of US equities. We evaluate and compare the selection methods

between themselves and against a random benchmark. Furthermore, we evaluate and compare the returns, risks and efficiency of the portfolios created as a result of the selections made by the machine learning algorithms;

- Chapter 5 provides a brief conclusion to the research, presents the several limitations and proposes future work to be carried out in the application of machine learning techniques to the active management of financial asset portfolios.

Chapter 2

Machine Learning

Machine learning is the design of algorithms that allow us to automatically extract patterns from data. From these, we can either make predictions on new data points or gain knowledge about a phenomenon (Deisenroth et al., 2020). There are three main components to any machine learning algorithm: data, a model and a learning procedure. First, we need a data set $\mathcal{D}$. This data set is represented in tabular form where each row denotes an example and each column represents an attribute. The model is a predictive function that aims to approximate an output value for a given input vector. For any new input vector the goal of this function is to output a value that captures the relationship that exists between the two. The form of this output will vary according to the problem we are considering. To aid the model in capturing the relationships, our goal will be to find the parameters of the considered function such that in the presence of new data, the predictive model will perform well. To do this, each model is accompanied by an optimization problem where we try to find the optimum of an objective function that we denote $\mathcal{L}$. This objective function will vary depending on the problem we consider.

2.1 Supervised vs. Unsupervised Learning

In the context of this thesis, we shall only consider two distinct groups of machine learning algorithms: supervised and unsupervised learning models (J. Friedman et al., 2001). In the case of **supervised learning**, we have data $\mathcal{D}$ that take the form of a set of N observations $(\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_\mathrm{n}, y_n) | n = 1, \ldots, N\}$ where y_n is the n^{th} output and the vector $\mathbf{x}_\mathrm{n}$ is the n^{th} input of D dimensions. We shall call the elements of this vector the

inputs or features. The objective of supervised learning is to find a function $f : \mathbb{R}^D \longrightarrow \mathbb{R}$ that allows us to approximate the relationship between the input and output variables. In doing so, we will be able to make predictions on new data points not observed in the past. If the model is capable of doing this accurately, it is said to have generalized well.

The vector $\mathbf{x}_\mathrm{n}$ is composed of continuous or categorical variables. As for y_n, if it is a real scalar, we are dealing with a **regression problem**. Otherwise, if it is a categorical variable coming from a finite set $y_n \in \{1, \ldots, C\}$, we are dealing with a **classification problem**. In financial application, an example of a regression problem would be to predict the future returns of a financial asset as a function of current market conditions. On the other hand, in a binary classification problem (where $C = 2$), the problem would be to predict whether or not the considered financial asset will have a higher future return than a benchmark over a given period. In the context of this thesis, when performing supervised learning, we will only consider binary classification problems. We are interested in whether an event will happen or not. We will encode this in the form of an indicator function:

$$y_n = \mathbb{1}_{\{\cdot\}} := \begin{cases} 1 \text{ if a condition is met} \\ 0 \text{ otherwise} \end{cases} \tag{2.1}$$

In the case of **unsupervised learning**, we again have access to a data set $\mathbf{X} = \{(\mathbf{x}_\mathrm{n})|n = 1, \ldots, N\}$, this time, however, without the corresponding output y_n. The objective of unsupervised learning is to discover interesting patterns in the data. Most often, what we discover with these methods is the presence of clusters in the data. Another common application of these algorithms is to compress data as a means to reducing its dimensionality. This has the advantage of allowing faster training times for supervised learning or to visualize a dataset in two dimensions that would otherwise be multidimensional. In finance, a clustering exercise would be to group different assets together according to a distance measure like the correlation coefficient (Pearson, 1895). A compression algorithm would allow us to keep a large part of the information contained in a company's financial results without having to use each and every valuation metric. For this thesis, we shall consider both clustering and the reduction of the dimensionality of our data.

2.2 Supervised Learning Algorithms (Classification)

2.2.1 Logistic Regression

We start by presenting the logistic regression model which will be the building block for the rest of the theory of supervised learning until neural networks. The aforementioned sections will closely follow the lecture notes by Jaggi and Flammarion (2020).

Logistic regression assumes a linear relationship between $\mathbf{x}_\mathrm{n}$ and y_n. Thus, one could write the model as:

$$\begin{aligned} y_n \approx f(\mathbf{x}_\mathrm{n}) &:= w_0 + w_1 x_{n1} + \cdots + w_D x_{nD} \\ &= w_0 + {\mathbf{x}_\mathrm{n}}^T \mathbf{w} \end{aligned} \tag{2.2}$$

where $\mathbf{w}$ is the vector of weights and w_0 is an additional bias term. In order to shorten this notation, we augment the vector $\mathbf{x}_\mathrm{n}$ to $[1, x_{n1}, \ldots, x_{nD}]^T$ and the vector $\mathbf{w}$ to $[w_0, w_1, \ldots, w_D]^T$ to get $y_n \approx f(\mathbf{x}_\mathrm{n}) := {\mathbf{x}_\mathrm{n}}^T \mathbf{w}$. However, in this approach, the outputs take values in $(-\infty, \infty)$, we thus transform it into outputs that take values in $(0, 1)$. We interpret this as the probability of a sample belonging to a class. The logistic/sigmoid function is used to make this transformation:

$$S(x) = \frac{e^x}{1 + e^x} \tag{2.3}$$

The final step in binary classification is to quantize the output:

$$\hat{f}^{(LR)}(\mathbf{x}_\mathrm{n}) = \mathbb{1}_{\left\{S({\mathbf{x}_\mathrm{n}}^T \mathbf{w}) \geq \frac{1}{2}\right\}} \tag{2.4}$$

We thus predict the probability of a new vector $\mathbf{x}_{new}$ belonging to a class as:

$$\begin{aligned} \mathbb{P}(1|\, \mathbf{x}_\mathrm{n}, \mathbf{w}) &= S\left(\mathbf{x}_{new}^T \mathbf{w}\right) \\ \mathbb{P}(0|\, \mathbf{x}_\mathrm{n}, \mathbf{w}) &= 1 - S\left(\mathbf{x}_{new}^T \mathbf{w}\right) \end{aligned} \tag{2.5}$$

Our objective with logistic regression is to estimate the vector of weights $\mathbf{w}$ that maximizes the probability of the set of observations being generated by the model. Using the fact that our dataset consists of independent pairs of $(\mathbf{x}_\mathrm{n}, y_n)$, we chose to maximize the likelihood L that the observations have been generated by the model

$$\max_{\mathbf{w}} L = \max_{\mathbf{w}} \prod_{n=1}^{N} S\left({\mathbf{x}_\mathrm{n}}^T \mathbf{w}\right)^{y_n} \left[1 - S\left({\mathbf{x}_\mathrm{n}}^T \mathbf{w}\right)\right]^{(1-y_n)} \tag{2.6}$$

For convenience, we maximize the logarithm of the likelihood to replace the product with a sum and invert the sign to turn the maximisation problem into a minimisation one. Our objective function then becomes:

$$\begin{aligned}\min_{\mathbf{w}} \mathcal{L}^{LR}(\mathbf{w}) &= \min_{\mathbf{w}} -\sum_{n=1}^{N} y_n \ln S\left(\mathbf{x_n}^T \mathbf{w}\right) + (1-y_n) \ln\left[1 - S\left(\mathbf{x_n}^T \mathbf{w}\right)\right] \\ &= \min_{\mathbf{w}} \sum_{n=1}^{N} \ln\left[\exp(\mathbf{x_n}^T \mathbf{w}) + 1\right] - y_n \, \mathbf{x_n}^T \mathbf{w}\end{aligned} \tag{2.7}$$

We select the vector of weights $\mathbf{w}^*$ such that:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{L}^{LR}(\mathbf{w})$$

To find $\mathbf{w}^*$, we consider the stationary points of the objective function by calculating its gradient, equating it to 0 and then solving for $\mathbf{w}$. However, there is no analytical solution for this problem. We therefore use an iterative algorithm: the mini-batch stochastic gradient descent. Starting from initial values, this algorithm randomly selects a subset $B \in [N]$ of the training data and takes a step in the opposite direction of the gradient until convergence. The weight update rule can thus be written as:

$$\mathbf{w}^{(i+1)} := \mathbf{w}^{(i)} - \gamma \frac{1}{|B|} \sum_{n \in B} \nabla \mathcal{L}_n^{LR}(\mathbf{w}^{(i)}) \tag{2.8}$$

where γ is the learning rate, and $\nabla \mathcal{L}^{LR}\left(\mathbf{w}^{(i)}\right)$ is the gradient of the loss function $\mathcal{L}^{LR}$ with respect to the weights $\mathbf{w}$. In the case of the negative log-likelihood:

$$\begin{aligned}\nabla \mathcal{L}^{LR}(\mathbf{w}) &= \sum_{n=1}^{N} \mathbf{x_n}\left(S(\mathbf{x_n}^T \mathbf{w}) - y_n\right) \\ &= \mathbf{X}^T\left[S(\mathbf{X}\mathbf{w}) - \mathbf{y}\right]\end{aligned}$$

2.2.2 Polynomial Logistic Regression

As mentioned, logistic regression assumes a linear relationship between $\mathbf{x_n}$ and y_n; this can be limiting when we are dealing with data that contains more complex, non-linear relationships. There is, however, a trick that we can use to augment our $\mathbf{x}_n$ vector such that logistic regression can now capture non-linear relationships. A common example of such a transformation would be to augment the data matrix by a polynomial basis. Consider one feature $\mathbf{x}_d$ of the data matrix $\mathbf{X}$:

$$\mathbf{x}_d = [x_{1,d}, x_{2,d}, \ldots, x_{N,d}]^T, \forall d = 1, \ldots, D \tag{2.9}$$

The augmented polynomial basis of degree M of this vector is:

$$\boldsymbol{\phi}(x_{n,d}) = \left[x_{n,d}, x_{n,d}^2, \ldots, x_{n,d}^M\right], \forall n = 1, \ldots, N \tag{2.10}$$

We apply this to each entry of our feature vector and then train a logistic regression model on this new augmented vector. The output can therefore be written:

$$y_n \approx S\left(\boldsymbol{\phi}(\mathbf{x}_\mathrm{n})^T \mathbf{w}\right) \tag{2.11}$$

The above is then quantized in the same way as logistic regression to get $\hat{f}^{(PLR)}$.

2.2.3 Underfitting and Overfitting

In the above description, we saw that it was possible to enrich our model by augmenting our data matrix by a polynomial basis of degree M. The question we ask ourselves is what degree M should be used? This value is a hyper-parameter that we will have to select appropriately for each problem we tackle. In this thesis, the majority of the models we shall consider will have hyper-parameters we must choose. The values of these must be selected in such a way that we have a model that is complex enough to capture the relationships that exist between the data but also robust enough so that we do not fit to the noise of the data (Figure 2.1). This is especially critical when dealing with financial data-sets as they tend to have low signal to noise ratios (Israel et al., 2020).

Formally, our data $\mathcal{D}$ follows a distribution $\mathcal{S}$ that we do not know. We train a model $f_{\mathcal{D},\Theta}$ on our data with parameters Θ and wish to assess its generalization capability. To do so, we are interested in the expected error:

$$\mathbb{L}_\mathcal{S}(f) = \mathbb{E}_\mathcal{S}\left[\ell(y, f(\mathbf{x}))\right] \tag{2.12}$$

where $\ell(-,-)$ is the error between the ground truth y and the predicted value $f(\mathbf{x})$. However, as we do not know the distribution $\mathcal{S}$, we therefore have to approximate this quantity empirically with the training error:

$$\mathbb{L}_\mathcal{D}(f_{\mathcal{D},\Theta}) = \frac{1}{|\,\mathcal{D}\,|} \sum_{(y_n, \mathbf{x}_\mathrm{n}) \in \mathcal{D}} \ell(y_n, f_{\mathcal{D},\Theta}(\mathbf{x}_n)) \tag{2.13}$$

It is nevertheless a mistake to measure the performance of the model on the same data that was used for training as in doing so, we do not guarantee the ability of the model to generalize to new data that follows the same distribution but has not been seen in the

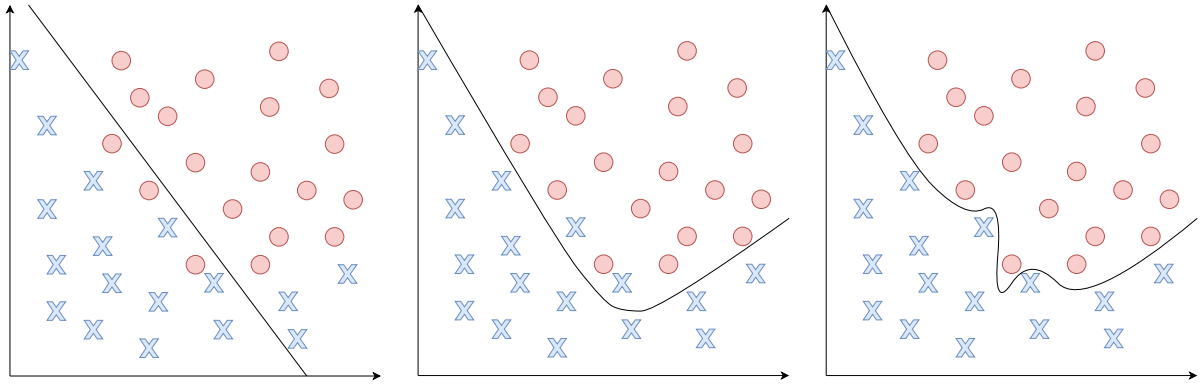

Figure 2.1: *Overfitting illustration. The graph shows a scenario where we have two features X1 and X2 and our goal is to find an adequate separation of the two classes which are crosses and circles. The first graph (left) shows an instance where the polynomial basis is 1. This is a classical logistic regression. We can see that there are a lot of points that are not well classified. In this case, our dataset has a non-linear relationship that a simple logistic regression cannot address. This is a case of underfitting. On the other hand, if we consider the graph on the right where M takes a large value, we can see that each point is classified correctly, however the curve performing the classification seems to be noisy, so it is natural to assume that it would not make good predictions on new points that have not been observed before. This is a case of overfitting. The plot in the center shows ideal case, the classification is not perfect but we have found a curve that robustly separates the points. The model is thus said to have generalized well.*

optimization process. When $\mathbb{L}_{\mathcal{D}}(f_{\mathcal{D},\Theta})$ is much smaller than $\mathbb{L}_{\mathcal{S}}(f_{\mathcal{D},\Theta})$ we have over-fitted to our data.

A simple approach to solve this problem and assess the generalization capability is to separate our data into two sets: one for training we denote $\mathcal{D}_{train}$ and the other for validation we denote $\mathcal{D}_{test}$. An approximation of the generalization error is then computed as:

$$\mathbb{L}_{\mathcal{D}_{test}}(f_{\mathcal{D}_{train},\Theta}) = \frac{1}{\mid \mathcal{D}_{test} \mid} \sum_{(y_n,\mathbf{x}_n)\in\mathcal{D}_{test}} \ell(y_n, f_{\mathcal{D}_{train},\Theta}(\mathbf{x}_n)) \tag{2.14}$$

An approach that is more suitable for financial time series because it allows us to evaluate out-of-sample performance over several periods is the K-fold purged time-series cross-validation (De Prado, 2018). To do this, we start by training over a fixed period of time, then we leave a gap and compute the validation error over a later period, also fixed, which comes after the last training observation and the gap period. We then consider the next period in a sliding window fashion and repeat the process K times. An illustration of this

is shown in Figure 2.2. The approximation of the generalization error is then the average of the validation error over all the K folds. For each model with hyper-parameters Θ, we choose the ones that minimize the approximation of the generalization error. This procedure of finding Θ is referred to as tuning the model.

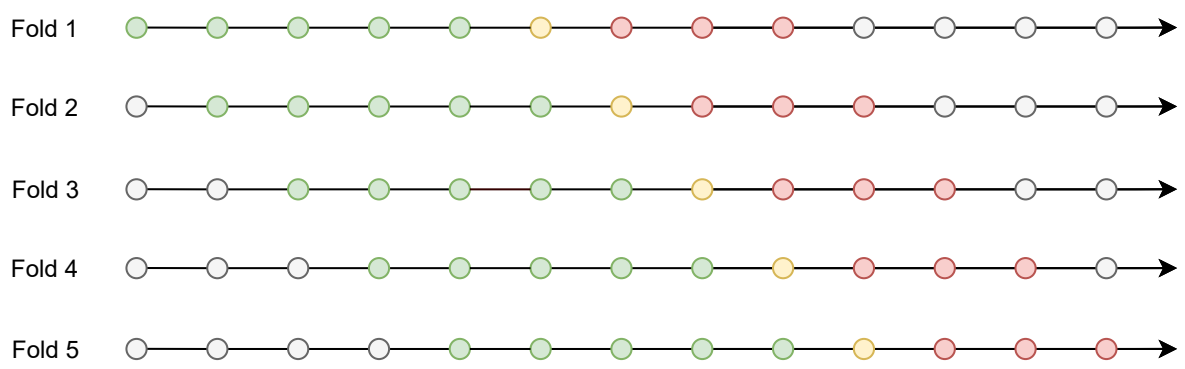

Figure 2.2: *K-fold purged time-series cross-validation. The graph shows the cross-validation procedure where we consider K=5 folds. We illustrate the training periods in green, the gap periods in orange and the test periods in red. We consider here that we have 13 years of data and each point on the line represents a year. In the first iteration, we train our model on the first 5 years, then we wait for 1 year before evaluating the model on the next 3 years. For the next iteration, we shift the process by one year and repeat the same operation. Thus, we finally have out-of-sample performance measures spanning 7 years. The average performance over this out-of-sample period is the estimate of the generalization error.*

2.2.4 Regularized Logistic Regression

To further reduce the possibility of over-fitting, we can add a penalty to our objective function in order to sanction models that are too complex where we would observe high values for the entries of our weight vector. In doing so, we modify the objective function to be:

$$\min_{\mathbf{w}} \mathcal{L}^{RLR}(\mathbf{w}) = \min_{\mathbf{w}} \mathcal{L}^{LR}(\mathbf{w}) + \Omega(\mathbf{w}) \tag{2.15}$$

where $\Omega(\mathbf{w})$ is a penalty term that measures the complexity of the model. We consider three types of regularization: ridge (L2), lasso (L1) and elastic net (EN). These can we

denoted by:

$$\begin{aligned}\Omega^{L_2}(\mathbf{w}) &= \frac{\lambda}{2}||\mathbf{w}||_2^2 \\ \Omega^{L_1}(\mathbf{w}) &= \lambda||\mathbf{w}||_1 \\ \Omega^{EN}(\mathbf{w}) &= \frac{\lambda_2}{2}||\mathbf{w}||_2^2 + \lambda_1||\mathbf{w}||_1\end{aligned} \tag{2.16}$$

where λ is a hyper-parameter that controls the strength of the regularization that we find through cross-validation. Again, the mini-batch stochastic gradient descent algorithm is employed to find the parameters that minimize the objective function. We express the gradients of each of the penalty functions as:

$$\begin{aligned}\nabla\Omega^{L_2}(\mathbf{w}) &= \lambda\mathbf{w} \\ \nabla\Omega^{L_1}(\mathbf{w}) &= \lambda\text{sgn}(\mathbf{w}) \\ \nabla\Omega^{EN}(\mathbf{w}) &= \lambda\mathbf{w} + \lambda\text{sgn}(\mathbf{w})\end{aligned} \tag{2.17}$$

2.2.5 Support Vector Classifier

Another approach, which aims to perform a more robust classification is the support vector classifier (SVC). For the presentation of this method, we consider data pairs $(\mathbf{X}, \tilde{\mathbf{y}}) = \{(\mathbf{x}_n, \tilde{y}_n)|n = 1, \ldots, N\}$ with $\tilde{y}_n \in \{-1, 1\}, \forall n$. In order to push the classifier to make more confident predictions, we minimize the hinge loss function:

$$\min_{\mathbf{w}} \mathcal{L}^{SVC}(\mathbf{w}) = \min_{\mathbf{w}} \sum_{n=1}^{N} \max\left[0, 1 - \tilde{y}_n \mathbf{x}_n^T \mathbf{w}\right] + \frac{\lambda}{2}||\mathbf{w}||_2^2 \tag{2.18}$$

As such, since the hinge loss is only equal to 0 when $\mathbf{x}_n^T\mathbf{w}$ is the same sign as $\tilde{y}_n$ and larger than 1, in order not to suffer any loss, the model must not only predict the sign correctly but the value of $\mathbf{x}_n^T\mathbf{w}$ must also be high enough. This can be considered as a more confident prediction. The hinge loss function is convex and has a sub-gradient. We can thus find $\mathbf{w}*$ which minimizes this function with the mini-batch stochastic sub-gradient descent algorithm. The sub-gradient of $\mathcal{L}^{SVC}(\mathbf{w})$ is:

$$\nabla\mathcal{L}^{SVC}(\mathbf{w}) = \begin{cases} -\tilde{y}_n\mathbf{x}_n + \lambda\mathbf{w} \text{ if } 1 - \tilde{y}_n\mathbf{x}_n^T\mathbf{w} > 0 \\ \lambda\mathbf{w} \text{ otherwise} \end{cases} \tag{2.19}$$

In the case where we have augmented the input vector enough such that $D > N$, we are left to try to find a potentially very large number of values for $\mathbf{w}^*$ that minimize the objective function. In this case, it is convenient to consider the formulation of the objective function in its dual form. As such, we will optimize for a vector of parameters which is of dimension N; thus allowing a more efficient implementation. The optimization problem in Equation 2.18 can be reformulated using convex duality (Kuhn & Tucker, 2014). By introducing $\alpha \in [0, 1]$, we replace $\max(0, 1 - yz)$ with $\max_{\alpha \in [0,1]} \alpha(1 - yz)$ to get:

$$\min_{\mathbf{w}} \mathcal{L}^{SVC}_{DUAL}(\mathbf{w}) = \max_{\alpha \in [0,1]^N} \min_{\mathbf{w}} \sum_{n=1}^{N} \alpha_n \left(1 - \tilde{y}_n \mathbf{x}_n^T \mathbf{w}\right) + \frac{\lambda}{2} ||\mathbf{w}||_2^2 \tag{2.20}$$

The gradient of the dual w.r.t $\mathbf{w}$ is:

$$\nabla \mathcal{L}^{SVC}_{DUAL}(\mathbf{w}) = -\sum_{n=1}^{N} \alpha_n \tilde{y}_n \mathbf{x}_n + \lambda \mathbf{w} \tag{2.21}$$

Equating to 0 and solving for $\mathbf{w}$ we get:

$$\begin{aligned} \mathbf{w}* &= \frac{1}{\lambda} \sum_{n=1}^{N} \alpha_n \tilde{y}_n \mathbf{x}_n \\ &= \frac{1}{\lambda} \mathbf{X}^T \tilde{\mathbf{Y}} \boldsymbol{\alpha} \end{aligned} \tag{2.22}$$

where $\tilde{\mathbf{Y}} := diag(\mathbf{y})$. Plugging $\mathbf{w}*$ in Equation 2.20 allows us to write:

$$\begin{aligned} \min_{\mathbf{w}} \mathcal{L}^{SVC}_{DUAL}(\mathbf{w}) &= \max_{\alpha \in [0,1]^N} \sum_{n=1}^{N} \alpha_n \left(1 - \frac{1}{\lambda} \tilde{y}_n \mathbf{x}_n^T \mathbf{X}^T \tilde{\mathbf{Y}} \boldsymbol{\alpha}\right) + \frac{\lambda}{2} \left|\left| \frac{1}{\lambda} \mathbf{X}^T \tilde{\mathbf{Y}} \boldsymbol{\alpha} \right|\right|_2^2 \\ &= \max_{\alpha \in [0,1]^N} \boldsymbol{\alpha}^T \mathbb{1} - \frac{1}{2\lambda} \boldsymbol{\alpha}^{\boldsymbol{T}} \boldsymbol{Q} \boldsymbol{\alpha} \end{aligned} \tag{2.23}$$

where $\boldsymbol{Q} := \tilde{\mathbf{Y}} \mathbf{X} \mathbf{X}^T \tilde{\mathbf{Y}}$. To find $\boldsymbol{\alpha}*$ that maximises Equation 2.23, we use the stochastic dual coordinate ascent algorithm (Shalev-Shwartz & Zhang, 2013). For a given function, we select at each iteration a random coordinate $n \in [N]$ and we optimize the objective function of Equation 2.23 with respect to this coordinate only. We are now interested in the sub-problem where we try to optimize a single variable. This sub-problem is denoted by:

$$\begin{aligned} &\max_{\gamma \in \mathbb{R}} f(\boldsymbol{\alpha} + \gamma \boldsymbol{e}_n) \\ &\text{s.t } 0 \leq \alpha_n + \gamma \leq 1 \end{aligned} \tag{2.24}$$

where $\boldsymbol{e}_n$ is a vector of zeros except at position n and $f(\boldsymbol{\alpha} + \gamma \boldsymbol{e}_n)$ is:

$$\begin{aligned} f(\boldsymbol{\alpha} + \gamma \boldsymbol{e}_n) &= (\boldsymbol{\alpha} + \gamma \boldsymbol{e}_n)^T \mathbb{1} - \frac{1}{2\lambda}(\boldsymbol{\alpha} + \gamma \boldsymbol{e}_n)^T \boldsymbol{Q}(\boldsymbol{\alpha} + \gamma \boldsymbol{e}_n) \\ &= \boldsymbol{\alpha}^T - \frac{1}{2\lambda}\boldsymbol{\alpha}^T \boldsymbol{Q}\boldsymbol{\alpha} - \frac{\gamma^2}{2\lambda}\boldsymbol{Q}_{nn} + \gamma\left(1 - \frac{1}{\lambda}\boldsymbol{\alpha}^T \boldsymbol{Q}\boldsymbol{e}_n\right) \end{aligned} \tag{2.25}$$

Computing the derivative of the above, equating to 0 and solving for γ, we get:

$$\gamma^* = \frac{\lambda}{\boldsymbol{Q}_{nn}}\left(1 - \frac{1}{\lambda}\boldsymbol{\alpha}^T \boldsymbol{Q}\boldsymbol{e}_n\right) \tag{2.26}$$

where $\boldsymbol{Q}_{nn} := \mathbf{x}_n^T \mathbf{x}_n y_n^2$ and $\boldsymbol{\alpha}^T \boldsymbol{Q}\boldsymbol{e}_n := \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_n y_n$. The update step to take at each iteration i of stochastic coordinate ascent is thus

$$\alpha_n^{(i)} = \min\left\{\max\left\{\alpha_n^{(i-1)} + \gamma^*, 0\right\}, 1\right\} \tag{2.27}$$

note that the min and max above are to ensure $\boldsymbol{\alpha} \in [0,1]^N$.

One of the appeals of the dual formulation is that the data is in the form of a kernel matrix $K = \mathbf{X}\mathbf{X}^T$. This allows us to augment the feature space very efficiently using a kernel function $\kappa(\mathbf{x}, \mathbf{x}')$. In this thesis, we consider the radial basis function as kernel:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\rho^2}\right) \tag{2.28}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are two samples and ρ is a hyper-parameter we need to find through cross-validation. The model prediction is then:

$$\hat{f}^{SVC}(\mathbf{x}_\mathrm{n}) = sgn\left(\sum_{n=1}^{N} \kappa(\mathbf{x}, \mathbf{x}_\mathrm{n})\boldsymbol{\alpha}_n\right) \tag{2.29}$$

2.2.6 Neural Network Classifier

Neural networks are powerful universal function approximators. It's structure can be described by looking at Figure 2.3. The first layer of nodes on the far left of the figure represents the input layer of size D. Each node present in this layer represents an element of the vector $\mathbf{x}_\mathrm{n}$. We then observe on the far right the output layer which will in our case be a real number between 0 and 1. As we are in the case of a binary classification, we shall once again try to predict the probability that we are in the class 0 or 1. The layers between the input and output layer are the so-called hidden layers. There are L hidden layers, each of size K. Note that even though we illustrate each hidden layer as having the same size, this is not necessarily the case in practice. Each node in the hidden layers

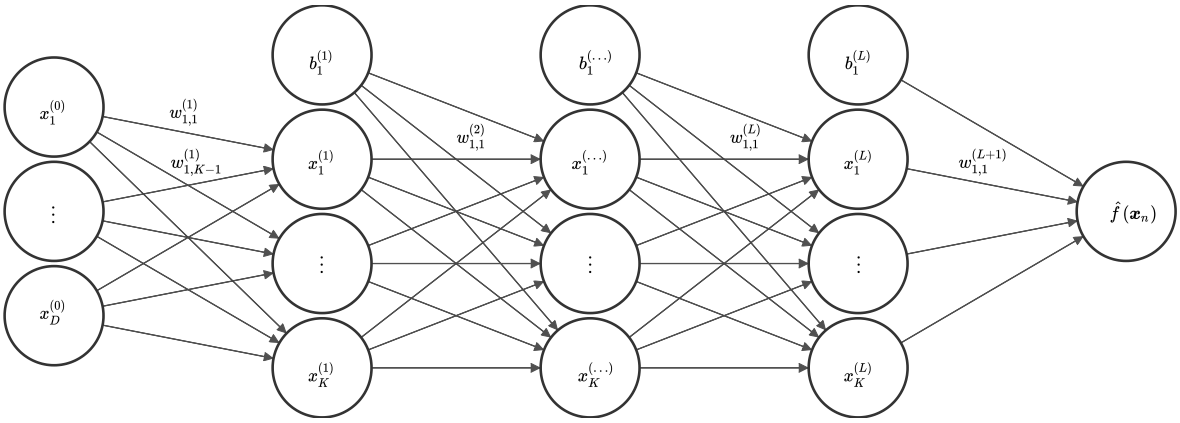

Figure 2.3: *Structure of the Neural Network. The nodes on the far left represents the input features of dimension D. The layer on the far right represents the output of the neural network which is a scalar. What lies between the two are the so-called hidden layers of dimension K.*

are connected to every node in the previous layer by a weighted edge. The value of node k in layer l is therefore denoted by:

$$x_k^{(l)} = \phi\left(\sum_i w_{i,k}^{(l)} x_i^{(l-1)} + b_k^{(l)}\right) \tag{2.30}$$

where $w_{i,k}^{(l)}$ is the edge from node i in layer $l-1$ to the node k in layer l. The function ϕ is the ELU activation function (Clevert et al., 2015) with $\alpha = 1$:

$$\phi(z) = \begin{cases} z \text{ if } z > 0 \\ \alpha(e^z - 1) \text{ otherwise} \end{cases} \tag{2.31}$$

The idea of the neural network is that we start with data as we provided it. Then we perform multiple weighted sums of our features and apply a non-linear transformation to them. The wish is that once we get to the final hidden layer L, our data is now transformed and represented in such a way that a simple logistic regression is sufficient to perform a highly accurate classification. As for the rest of the models, we are therefore looking for the weights and bias parameters that minimize the loss function. For this, we will use stochastic gradient descent. As we are dealing here with a model that has many parameters, we modify the notation such that we can compute the gradients simultaneously. We thus compute for each layer $l = 1, \dots, L+1$:

$$\begin{aligned} \mathbf{z}^{(l)} &= \left(\mathbf{W}^{(l)}\right)^T \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)} \\ \mathbf{x}^{(l)} &= f^{(l)}(\mathbf{x}^{(l-1)}) = \phi(\mathbf{z}^{(l)}) \end{aligned} \tag{2.32}$$

with $\mathbf{x}^{(0)} = \mathbf{x}_n$. where $\mathbf{W}^{(l)}$ is the matrix of weights that go from layer $l-1$ to layer l and $\mathbf{b}^{(l)}$ a vector of the bias terms. As a result, the overall function is a composition of the above:

$$f(\mathbf{x}^{(0)}) = f^{(L+1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(\mathbf{x}^{(0)}) \tag{2.33}$$

To use the stochastic gradient descent algorithm, we are thus interested in computing the gradient of the loss function w.r.t the weights and biases. In the case of the cross-entropy loss function:

$$\mathcal{L}_n^{(NN)} = -y_n \ln\left[\hat{f}\left(\mathbf{x}^{(0)}\right)\right] - (1-y_n)\ln\left[1-\hat{f}\left(\mathbf{x}^{(0)}\right)\right] \tag{2.34}$$

and logistic activation function (Equation 2.3) for the output layer, we set

$$\delta^{(L+1)} = \frac{y_n - \mathbf{x}^{(L+1)}}{\mathbf{x}^{(L+1)}\left(1-\mathbf{x}^{(L+1)}\right)} S'(z^{(L+1)}) \tag{2.35}$$

and compute:

$$\delta^{(l)} = \left(\mathbf{W}^{(l+1)}\delta^{(l+1)}\right) \odot \phi'(\mathbf{z}^{(l)}), \forall l = L, \ldots, 1 \tag{2.36}$$

where $S'(z) := S(z)\left(1-S(z)\right)$ is the derivative of the logistic function, $\phi'(z) := \alpha e^z \mathbb{1}_{\{z<=0\}} + \mathbb{1}_{\{z>0\}}$ is the derivative of the ELU activation function and $\odot$ is the element-wise product. Then, we can compute the gradients of the loss function with respect to the model parameters with:

$$\begin{aligned} \frac{\partial \mathcal{L}_n}{\partial w_{i,j}^{(l)}} &= \delta_j^{(l-1)} \mathbf{x}_i^{(l-1)} \\ \frac{\partial \mathcal{L}_n}{\partial b_j^{(l)}} &= \delta_j^{(l)} \end{aligned} \tag{2.37}$$

While neural networks provide state-of-the-art results in many disciplines (Zhai et al., 2021). Once the transformations are applied to the input features, we are not able to interpret the mapping performed by the network to arrive at the predictions. Moreover, in noisy environments such as financial applications, these models tend to overfit to the data.

2.2.7 Decision Tree Classifier

Predictions made by decision trees are easily interpretable and traceable. This coupled with its flexibility makes it one of the most popular models in some industries (Podgorelec et al., 2002) and represents the model on which two of the subsequent models will be

based on. We present classification trees in this section as introduced by Breiman et al. (1984) in their seminal work. A classification tree partitions the feature space $\mathbf{X}$ into disjoint subsets where for each subset, the output variable is modelled as a constant.

We illustrate the structure of said tree in Figure 2.4 where we consider two features x_1 and x_2. The input space is first split based on the condition $x_1 \leq w_1$. The resulting two subsets are consequently split recursively in the same fashion until a stopping criteria is met. We denote the final regions $\mathcal{R}_m$ as the m^{th} terminal nodes. To make predictions on a new input vector $[x_1', x_2']^T$, we would follow the path of the tree that satisfies the w_i conditions. The constant belonging to the terminal node in which the sample lies will then be predicted.

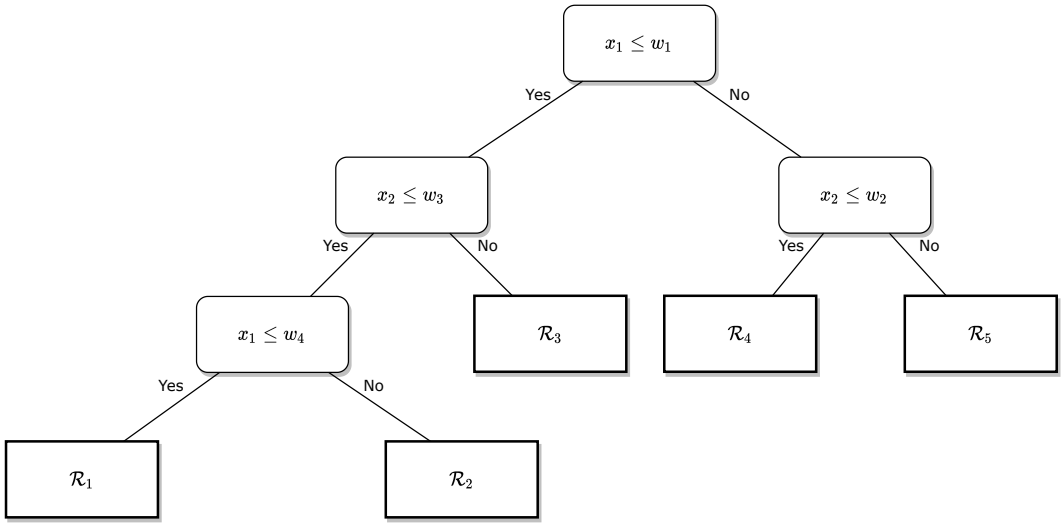

Figure 2.4: *Illustration of a decision tree. The splitting begins at the root node. The initial split partitions the data according to whether* $x_1 \leq w_1$*. This creates two new data sets; the first is partitioned again by checking the condition* $x_2 \leq w3$*. As one of these new partitions contains only one class, which is in the region* $\mathcal{R}_3$ *the splitting stops for that branch. The other branch is partitioned once more to create the regions* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *based on whether* $x_1 \leq w_4$*. On the other side, after the first partition is done, the regions* $\mathcal{R}_4$ *and* $\mathcal{R}_5$ *are created from the condition* $x_2 \leq w_2$*. This procedure is repeated each time a new prediction has to be performed.*

The unique features $\mathbf{x}_d, d = 1, \ldots, D$ and split points w_i are selected by iterating over all pairs $(\mathbf{x}_d, w_i)$ and picking the one that minimizes a given impurity measure. This is done by employing a greedy strategy; we look for a local optimum at the root node and

then repeat the process at each subsequent node. This allows us to obtain a solution that is usually good even though we are not guaranteed to have an optimal one. Thus at each node of the tree, we consider one feature of $\mathbf{X}$ that has q distinct values. We then order $\mathbf{x}_d$ and construct the constant:

$$w_i = \frac{x_{i+1,d} + x_{i,d}}{2}, \forall i = 1, \ldots, q-1 \tag{2.38}$$

We consider all combinations of $\mathbf{x}_d$ and w_i to create a split based on the condition $\mathbf{x}_d \leq w_i$ and compute the resulting impurity of making such a split. The cross-entropy $\mathcal{H}_m$ is used as a measure of impurity for a given node (J. Friedman et al., 2001):

$$\mathcal{H}_m = -\sum_{k=1}^{K} \hat{p}_{mk} \;\; ln\; \hat{p}_{mk} \tag{2.39}$$

where $\hat{p}_{mk}$ is the proportion of a class k in terminal node $\mathcal{R}_m$:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{\mathbf{x}_n \in \mathcal{R}_m} \mathbb{1}_{\{y_n = k\}} \tag{2.40}$$

with N_m the number of data samples in the region. The prediction is done by selecting the majority class of the region.

$$\hat{f}_m^{DT}(\mathbf{x}_n) = \arg\max_k \hat{p}_{mk} \tag{2.41}$$

We thus select at each node the optimal feature $\mathbf{x}_{d*}$ and constant w_i* that minimize the cross-entropy for both newly created subsets of $\mathbf{X}$. We denote by N_{mL} & N_{mR} the number of data samples in the newly created left and right sub-regions $\mathcal{R}_{m_L}$ & $\mathcal{R}_{m_R}$.

$$\mathcal{L}_m^{DT}(d, w) = \arg\min_{d,w} \left[\min \frac{1}{N_{m_L}} \sum_{\mathbf{X} \in \mathcal{R}_{m_L}} \mathcal{H}_{m_L} + \min \frac{1}{N_{m_R}} \sum_{\mathbf{X} \in \mathcal{R}_{m_R}} \mathcal{H}_{m_R} \right] \tag{2.42}$$

This procedure is repeated until the maximum depth is reached or if it is no longer possible to find a split that would reduce the impurity. Growing a tree such that this stopping criteria is met for each terminal node would ensure a perfect fit of the data. However, this predictor would probably contain a lot of noise and not generalize well; a sign of overfitting. Other stop criteria include a maximum allowable depth or a minimum number of samples to be required in a terminal node. These are selected through a cross-validation procedure.

2.2.8 Random Forest Classifier

The main concern with the decision tree classifier is that there is a high variance in the performance of the model. If the data changes a little bit, it can result in a significantly different tree and predictions. By combining the ideas of the bagging tree from Breiman (1996) and random feature splitting by Amit and Geman (1997): Breiman (2001) proposes random forests. The idea behind this algorithm is to train several decision trees; each on different random subsets of the data. Doing this allows us to reduce the variance of the predictor whilst keeping the same bias (Denuit et al., 2020).

The bagging tree algorithm fully grows multiple trees that are each trained on different random samples of the data. Then, the predictions of each tree are combined through a majority vote. The random forest algorithm extends this idea to further reduce the correlation between trees, at each node in the decision tree construction process, only v features from $\mathbf{X}$ with $v \leq D$ are randomly sampled and considered for the split. The number of features to consider at each split is selected through cross-validation. Finally, once all the trees are built, an average of the predictions of each tree is performed:

$$\hat{f}^{RF}(\mathbf{x}_{\mathrm{n}}) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{DT}_{\mathcal{D}^b}(\mathbf{x}_{\mathrm{n}}) \tag{2.43}$$

where B is the number of sampled subsets from the full data set $\mathcal{D}$. Equation 2.43 outputs a real scalar in (0,1) that can be interpreted as the confidence of the model. Indeed, if each decision tree predicts the same class, we will obtain a value of 1 or 0, indicating that the model is very confident in its prediction. We can then quantize this number in the same way as in Equation 2.4.

2.2.9 Gradient Boosting Tree Classifier

We present the gradient boosting algorithm as described by J. H. Friedman (2001). The idea behind this machine learning algorithm is also to combine the predictions of several classification trees. This time, however, we train the trees in a sequential manner where each new tree tries to correct the errors of the one that precedes it. We start by predicting a constant value for all data samples

$$\hat{f}_0^{GBCT}(\mathbf{x}_{\mathrm{n}}) = \arg\min_{\hat{y}} \sum_{n=1}^{N} \mathcal{L}(y_n, \hat{y}) \tag{2.44}$$

In the case of binary classification, we use cross-entropy loss function:

$$\mathcal{L}^{GBCT}\left(y_n, \hat{y}_n\right) = -y_n \ln\left(\hat{y}_n\right) - \left(1 - y_n\right) \ln\left(1 - \hat{y}_n\right) \tag{2.45}$$

to get the constant prediction:

$$\hat{f}_0^{GBCT}(\mathbf{x}_\mathrm{n}) = \frac{1}{N} \sum_{n=1}^{N} y_n \tag{2.46}$$

We then compute the pseudo-residuals $\tilde{y}_n$ which is the gradient of the loss function w.r.t the model predictions

$$\tilde{y}_n = \frac{\partial \mathcal{L}^{GBCT}\left(y_n, \hat{y}_n\right)}{\partial \hat{y}_n} = -\frac{y_n}{\hat{y}_n} + \frac{1 - y_n}{1 - \hat{y}_n}, \forall n \tag{2.47}$$

with this, at each iteration $i = 1, \ldots, I$; a decision tree is trained on a new pair $(\mathbf{X}, \tilde{\mathbf{y}})$ where $\tilde{\mathbf{y}}$ represents the pseudo-residuals of the preceding model. The predictions of a Gradient Boosting Classification Tree consisting of K trees can be written as:

$$\hat{f}^{GBCT}(\mathbf{x}_\mathrm{n}) = \sum_{i=1}^{I} \gamma \hat{f}_i^{DT}(\mathbf{x}_\mathrm{n}) \tag{2.48}$$

where γ is a constant learning rate obtained through cross-validation.

2.2.10 Stacking Classifier

We have so far presented a number of classification algorithms that each have their strengths and weaknesses. In order to try to combine these predictions to have a more robust classification, we consider another optimization problem where we use a meta-model to try to combine the classifications to compensate for classifier biases (Wolpert, 1992). Consider our data now consists of the pair $(\hat{\mathbf{Y}}, \mathbf{y}) = \{(\hat{\mathbf{y}}_n, y_n) | n = 1, \ldots, N\}$ where $\hat{\mathbf{Y}}$ is an $n \times m$ matrix of n predictions made by m models. As before y_n is the output variable and our ground truth. We seek to combine our forecasts by training a logistic regression model as in Equation 2.4. The final combined prediction thus becomes:

$$\hat{f}^{(COMB)}(\mathbf{x}_\mathrm{n}) = S\left(\hat{\mathbf{y}}_n^T \mathbf{w}\right) \tag{2.49}$$

With the objective of avoiding extreme weights assigned to each model, we restrict the values of $\mathbf{w} \in (0, 1)$. $\mathbf{w}^*$ is then found with projected stochastic gradient descent. Thus, at each iteration of the optimization procedure, we project the values of $\mathbf{w}$ back onto $(0, 1)$.

2.3 Unsupervised Learning Algorithms

2.3.1 Clustering

Louvain Algorithm

We perform the presentation of clustering algorithms with the Louvain method which aims at extracting a community structure from a large graph-like network (Blondel et al., 2008). A graph is a structure composed of nodes linked together by edges. We consider in this thesis a fully connected and undirected graph where each node is connected to another by a weighted edge. When performing community detection, we would like to obtain clusters of nodes that are strongly connected between them but only weakly connected with nodes in other clusters. The standard method used to solve such a task is the optimisation of modularity. Given a network with N nodes, we define the partition vector $\boldsymbol{\sigma}$ of dimension N where each entry σ_i represents the community to which node i belongs. Then, for this partition, modularity measures the fraction of edges that are in a community minus the expected fraction if the edges were randomly appointed to a cluster. This expectation thus represents a null model. The modularity Q of a given partition $\boldsymbol{\sigma}$ is given by:

$$Q(\boldsymbol{\sigma}) = \frac{1}{A_{tot}} \sum_{i,j} \left[A_{i,j} - \mathbb{E}[A_{ij}]\right] \delta(\sigma_i, \sigma_j) \tag{2.50}$$

where $\delta(\sigma_i, \sigma_j) := \mathbb{1}_{\{\ \sigma_i = \sigma_j\}}$, A_{ij} is the weight of the edge going from node i to node j, $A_{tot} := \sum_{i,j} A_{i,j}$ is a normalizing constant and the null model $\mathbb{E}[A_{ij}] := \frac{k_i k_j}{2m}$ with $k_i := \sum_j A_{i,j}$ the sum of the weights linked to node i and $2m := A_{tot}$. Since optimizing this objective function for a large graph is computationally infeasible, we turn to the Louvain method which allows us to find a fairly good solution in a very efficient way.

The algorithm works in two phases, initially we assign each node to its own community. Then, for each node i we consider its j-neighbors and calculate the change in modularity ΔQ that would occur if we removed i from its community and integrated it into the community B of one of its j-neighbors. The change in modularity is written as:

$$\Delta Q = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \tag{2.51}$$

where $\sum_{in}$ is the sum of the weights of the links inside the community B, $\sum_{tot}$ is the sum of the links incident to nodes in B, k_i is the sum of weights of links incident to node

i, $k_{i,in}$ is the sum of weights of links from i to nodes in B and m is the sum of weights of all links in the network.

Finally, we assign node i to the community that would allow the largest gain in modularity. We continue to traverse each node until there are no more changes in the nodes' attibutions to new groups. We then start phase 2 of the algorithm where we create a new graph where a node now represents a community found in the previous step. These new nodes are referred to as hyper-nodes. The weights of the edges are represented by the sum of all the links between the edges that are in the communities. Steps 1 and 2 are then repeated successively until there are no more changes.

Modified Louvain Algorithm

In the context of this thesis, our goal is to find communities in a correlation matrix that we denote $\mathbf{C}$ that has been constructed based on T observations of normalized returns of N assets. Being a symmetric matrix similar to the matrix $\mathbf{A}$ mentioned above it would be natural to replace A_{ij} in Equation 2.50 with C_{ij} and apply the community detection algorithm. However, while the null model mentioned in Equation 2.50 is suitable in the context of network analysis, MacMahon and Garlaschelli (2013) argue that it is not adequate for a correlation matrix computed on returns of financial assets as it introduces a bias. As such, modularity would not give greater importance to asset pairs that are highly correlated. Rather, it favors asset pairs that have a higher correlation compared to the correlations between the assets and the aggregate sum of all assets in the considered universe. MacMahon and Garlaschelli (2013) propose to redefine modularity as:

$$\tilde{Q}(\boldsymbol{\sigma}) = \frac{1}{\mathbb{V}ar[\mathbf{r}_{tot}]} \sum_{i,j} C_{ij}^{(f)} \delta(\sigma_i, \sigma_j) \tag{2.52}$$

where $C_{ij}^{(f)}$ is an entry of the filtered correlation matrix $\mathbf{C}^{(f)}$ defined in Equation 3.10 and $\mathbf{r}_{tot} := \{r_{tot}^{(1)}, \ldots, r_{tot}^{(T)}\}$ is a vector of returns of the aggregate sum of all the assets at time t with $r_{tot}^{(t)} := \sum_{n=1}^{N} r_n^{(t)}$ where $r_n^{(t)}$ the return of the n^{th} asset at time t. Therefore, the change in modularity if node i is removed from its community and put in the community of one of its j-neighbors is:

$$\Delta\tilde{Q} = \frac{C_{ij}^{(f)}}{\mathbb{V}ar[\mathbf{r}_{tot}]} \tag{2.53}$$

The added benefit of this correction is that assets within a cluster are strongly correlated while those outside are anti-correlated. This is of great interest when we use this method

to optimize a portfolio of assets.

2.3.2 Dimensionality Reduction

Principal Component Analysis

The objective of principal component analysis (PCA) is to reduce the dimensionality of our $(D \times N)$ data matrix $\mathbf{X}$ (Deisenroth et al., 2020). We want to project $\mathbf{X}$ to a matrix $\mathbf{C}$ of dimension $(K \times D)$ with $K << D$ to get a new $(K \times N)$ matrix $\tilde{\mathbf{X}}$ that retains most of the information contained in $\mathbf{X}$. To do so, we find the best $\mathbf{R}$ of dimension $(D \times K)$ such that when we try to reconstruct the matrix $\mathbf{X}$ from $\mathbf{CX}$, we have lost minimal amounts of information. We consider the following optimisation problem:

$$\min_{\mathbf{C},\mathbf{R}} \mathcal{L} = \min_{\mathbf{C},\mathbf{R}} ||\mathbf{X} - \mathbf{RCX}||_F^2 \tag{2.54}$$

where $||A||_F^2 := \sum_{i,j} |A_{i,j}|^2$ is the Frobenius norm. In order to solve this optimization problem, we refer to the following lemma (Eckart & Young, 1936):

Lemma 1. *For any $D \times N$ matrix $\mathbf{X}$ and any $D \times N$ rank-K matrix $\tilde{\mathbf{X}}$:*

$$||\mathbf{X} - \tilde{\mathbf{X}}||_F^2 \geq ||\mathbf{X} - \underbrace{\mathbf{U}_k}_{\mathbf{R}} \underbrace{\mathbf{U}_k^T}_{\mathbf{C}} \mathbf{X}||_F^2 = \sum_{i \geq K+1} s_i^2$$

where $\mathbf{X} = \mathbf{USV}^T$ is the Singular Value Decomposition (SVD) of $\mathbf{X}$, the s_i terms are the singular values of $\mathbf{X}$ and $\mathbf{U}_k$ is the $(D \times K)$ matrix of the first K columns of $\mathbf{U}$.

Since the columns of $\mathbf{U}$ are the left singular vectors of $\mathbf{X}$, the largest source of information is in the first singular vector, the second largest in the second, and so on. Therefore, to compress our data in an optimal way so as to retain a maximum of information when we go to K dimensions, we compute:

$$\tilde{\mathbf{X}} = \mathbf{U}_k^T \mathbf{X} \tag{2.55}$$

We select K so that $c\%$ of the variance of the data is retained. Using the diagonal of the $\mathbf{S}$ matrix from SVD, we pick K for which:

$$\frac{\sum_{k=1}^{K} S_{kk}}{\sum_{n=1}^{N} S_{nn}} \geq c\% \tag{2.56}$$

Chapter 3

Portfolio Optimization

3.1 Mean-Variance Portfolio

Portfolio optimization is one of the most essential issues in financial asset management. Asset managers select N financial assets in which they would like to invest and are then faced with the problem of optimally allocating ressources to these assets in order to best fulfill their utility function. Harry Markowitz studied the question and assuming that a rational investor prefers a less risky portfolio for a given level of return and a portfolio with a better return for a given level of risk (where risk is measured as the variance of the returns) defined the quadratic utility function of the investor as (Markowitz, 1952):

$$\mathcal{U}(\mathbf{w}) = \mathbf{w}^T \boldsymbol{\mu} - \frac{\phi}{2} \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w} \tag{3.1}$$

where $\mathbf{w} := [w_1, \ldots, w_N]^T$ is the vector of weights assigned to each asset in the portfolio, $\boldsymbol{\mu} := \mathbb{E}[\mathbf{R}]$ is the vector of expected returns with $\mathbf{R} := [\mathbf{r}_1, \ldots, \mathbf{r}_N]^T$ where $\mathbf{r}_n$ is the vector of returns of the n^{th} financial asset, $\boldsymbol{\Sigma} := \mathbb{E}[(\mathbf{R} - \boldsymbol{\mu})(\mathbf{R} - \boldsymbol{\mu})^T]$ is the covariance matrix of asset returns and $\phi \geq 0$ is a risk aversion parameter. The objective is thus to choose the weights that maximize the investor's utility. This problem is often re-written as the minimization of the portfolio variance under the constraint of a minimum level of return, full investment of the portfolio and bounds applied to the weight vector (e.g. not allowing

short selling):

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w} \\ \text{s.t} \quad & \mathbf{w}^T\boldsymbol{\mu} \geq \mu^* \\ & \mathbf{w}^T\mathbf{1}_N = 1 \\ & lb \leq \mathbf{w} \leq ub \end{aligned} \tag{3.2}$$

where μ^* is the minimum target return, lb is a lower bound, ub is the upper bound and $\mathbf{1}_N$ is a vector of ones of dimension N. The above minimization problem is solved using common quadratic programming software (Diamond & Boyd, 2016). As in practice we do not know $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, it is common to estimate them from the data. Therefore, we calculate $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ as follows:

$$\begin{aligned} \hat{\boldsymbol{\mu}} &= \frac{1}{T}\sum_{t=1}^{T}\mathbf{r}_t \\ \hat{\boldsymbol{\Sigma}} &= \frac{1}{T-1}\sum_{t=1}^{T}(\mathbf{r}_t - \hat{\boldsymbol{\mu}})(\mathbf{r}_t - \hat{\boldsymbol{\mu}})^T \end{aligned} \tag{3.3}$$

where T is the number of data samples we have and $\mathbf{r}_t$ is the vector of return of N assets at time t. These estimates however, have been shown to cause undesirable effects that we discuss hereafter.

3.2 Issues with Mean-Variance

While the presented method is optimal in-sample, several works (DeMiguel et al., 2009; Michaud, 1989; Jobson and Korkie, 1980) show that out of sample, this technique offers solutions that are often unstable and concentrated.

Indeed, according to Stein (2020) the use of the historical mean of returns in the estimation of the vector of expected returns is not admissible due to the multivariate nature of the problem. As a result, some investors may think of predicting expected returns by other more sophisticated means. Given the inherent noise and reflexivity of the markets, however, this exercise is very difficult to perform successfully and predictions will unavoidably be prone to estimation error. The main concern of this is that the optimization routine will latch on to the values that are most subject to these errors and

attribute extreme weights to them. For instance, it is typical to see large weights assigned to companies with large expected returns and vice versa.

This undesirable effect leads certain practitioners to forego the use of the expected returns vector in favor of minimizing the variance of the portfolio only. Nevertheless, given that the assumptions of temporal stationarity and linear interdependence of the covariance matrix is violated when estimated from data on asset returns, estimation errors are still present and lead to the same undesirable concentration effects. Here, the optimization routine tends to assign extreme weights to assets with negative correlations and small variances. This concentration, however, is less severe than for the approach using the expected vector of returns.

The second major concern of the mean-variance paradigm routine is that numerically, its solution is very unstable. Indeed, we observe large changes in the solutions when the input covariance matrix only slightly changes. This is due to the fact that the covariance matrix must be inverted in the optimization routine to reach an optimal solution. When this covariance matrix is ill-conditioned, i.e. has a high condition number, then the inversion is prone to large errors. De Prado (2018) shows that the more correlated the assets in the investment universe are, the higher the condition number. Hence, the mean-variance paradigm is the least effective when it is most needed.

These concerns regarding concentration and instability mean that in practice, when evaluated out of sample, an allocation that gives equal weight to each asset defined as:

$$\mathbf{w}^*_{naive} = \frac{1}{N}\mathbf{1}_N \tag{3.4}$$

obtain better performances.

As such, to reduce these concerns to a minimum, we only consider in this thesis the problem of minimizing the variance of the portfolio. In addition, we present three techniques that aim to address the sources of estimation errors and instability.

3.3 Shrinkage

Ledoit and Wolf, 2004, propose a new formula to estimate the covariance matrix by shrinking it to find a compromise between an unbiased estimate that is susceptible to estimation error denoted by $\mathbf{S}$ and a highly biased estimate $\mathbf{F}$ that contains little estimation error. Considering the sample covariance matrix as the former and the constant

model covariance matrix as the latter; they propose to replace the sample covariance matrix used in the optimization problem described in Equation 3.2 by:

$$\hat{\mathbf{\Sigma}}_{shrink} = \hat{\delta}^* \underbrace{\bar{\sigma}^2 \boldsymbol{I}_N}_{\mathbf{F}} + (1 - \hat{\delta}^*) \underbrace{\hat{\mathbf{\Sigma}}_{sample}}_{\mathbf{S}} \tag{3.5}$$

where $\bar{\sigma}^2 := \text{trace}(\mathbf{S})/N$, $\boldsymbol{I}_N$ is an identity matrix of size N and $\hat{\delta}^*$ is the optimal shrinkage constant defined by:

$$\hat{\delta}^* = \max\left\{0, \min\left\{\frac{\hat{\kappa}}{T}, 1\right\}\right\} \tag{3.6}$$

where $\hat{\kappa} := \frac{\hat{\pi} - \hat{\rho}}{\hat{\gamma}}$ with $\hat{\pi}, \hat{\rho}, \hat{\gamma}$ the consistent estimators of π, ρ, γ:

$$\begin{aligned}
\pi &= \sum_{i=1}^{N}\sum_{j=1}^{N} \text{AsyVar}\left[\sqrt{T}S_{ij}\right] \\
\rho &= \sum_{i=1}^{N}\sum_{j=1}^{N} \text{AsyCov}\left[\sqrt{T}F_{ij}, \sqrt{T}S_{ij}\right] \\
\gamma &= \sum_{i=1}^{N}\sum_{j=1}^{N} (F_{ij} - S_{ij})^2
\end{aligned}$$

3.4 Filtering

Another approach, based on Random Matrix Theory, is also commonly used in order to filter the covariance matrix in such a way as to try to remove as much noise as possible from the covariance matrix while retaining the signal (Potters et al., 2005). Throughout this section, we consider the empirical cross-correlation matrix $\hat{\mathbf{C}}$ and not the covariance matrix $\hat{\mathbf{\Sigma}}$, however, we can go from the latter to the former by performing:

$$\hat{\mathbf{C}} = \hat{\mathbf{\Sigma}} \oslash \boldsymbol{\sigma}\boldsymbol{\sigma}^T \tag{3.7}$$

where $\boldsymbol{\sigma} := \sqrt{diag(\hat{\mathbf{\Sigma}})}$ is a vector of standard deviations and $\oslash$ is the element-wise division. The theory mentions that for a correlation matrix composed of N standardized random time series of length T, the eigenvalues of this random matrix converge to a Marchenko–Pastur distribution (Götze & Tikhomirov, 2004) as $N, T \to \infty$ and $1 < T/N < +\infty$. The probability density function of this distribution is defined as:

$$f(\lambda) = \frac{T}{N}\frac{\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}}{2\pi\lambda}\mathbf{1}_{\{\lambda \in [\lambda_-, \lambda_+]\}} \tag{3.8}$$

where $\lambda_{\pm} := \left[1 \pm \sqrt{\frac{N}{T}}\right]^2$ represents the maximum and minimum expected eigenvalues.

Therefore, considering our empirical correlation matrix $\hat{\mathbf{C}}$, any eigenvalue that lies between λ_+ and λ_- can be considered as being due to noise. On the other hand, any eigenvalue higher than λ_+ can be considered as being due to signal. As such we can decompose the correlation matrix as:

$$\hat{\mathbf{C}} = \underbrace{\hat{\mathbf{C}}^{(n)}}_{\text{noise}} + \underbrace{\hat{\mathbf{C}}^{(s)}}_{\text{signal}} \tag{3.9}$$

where $\hat{\mathbf{C}}^{(n)} := \sum_{i:\lambda_i \leq \lambda_+} \lambda_i \mathbf{v}_i \mathbf{v}_i^T$ with $\mathbf{v}_i$ the eigenvector associated with the i^{th} eigenvalue and $\hat{\mathbf{C}}^{(s)} := \hat{\mathbf{C}} - \hat{\mathbf{C}}^{(n)}$. Among others, MacMahon and Garlaschelli (2013) observed that by performing this operation on an empirical correlation matrix formed on returns of stocks contained in the S&P500 index; the largest eigenvalue λ_m is much larger than all others. They therefore identify the eigenvector corresponding to the latter eigenvalue as the market component that influences all stocks in the index. As a result, they propose to further decompose the correlation matrix as follows:

$$\hat{\mathbf{C}} = \underbrace{\hat{\mathbf{C}}^{(n)}}_{\text{noise}} + \underbrace{\hat{\mathbf{C}}^{(m)}}_{\text{market}} + \underbrace{\hat{\mathbf{C}}^{(f)}}_{\text{filtered}} \tag{3.10}$$

where $\hat{\mathbf{C}}^{(m)} := \lambda_m \mathbf{v}_m \mathbf{v}_m^T$ and $\hat{\mathbf{C}}^{(f)} := \sum_{i:\lambda_+ < \lambda_i < \lambda_m} \lambda_i \mathbf{v}_i \mathbf{v}_i^T$. As such, we replace in the mean-variance optimization paradigm the sample covariance matrix with $\hat{\mathbf{\Sigma}}_{\text{filtered}}$ which we compute by re-ordering Equation 3.7 to convert $\hat{\mathbf{C}}^{(f)}$ back to a covariance matrix.

3.5 Nested Clustered Optimization

De Prado (2020) tackles the source of instability caused by certain structures of covariance matrices where clusters of correlated assets are present. To this end, the first step of Nested Clustered Optimization (NCO) consists in clustering into K groups the filtered correlation matrix $\hat{\mathbf{C}}^{(f)}$ using unsupervised machine learning algorithms. We use the modified Louvain method described in section 2.3.1 for this task. For each cluster $k = 1, \ldots, K$, we compute the filtered covariance matrix $\hat{\mathbf{\Sigma}}^{(k)}_{\text{filtered}}$ of the assets in the cluster and perform mean-variance optimization to get $\mathbf{w}_k^*$. We augment this vector by adding entries for each asset not included in the cluster and set their weights to 0. Thus, $\mathbf{w}_k^*$ is a vector of N dimensions (the number of assets) where each entry is the intra-cluster weight attributed to the n^{th} security and the stocks not in the cluster have a null weight.

This is repeated for each cluster to form the $(N \times K)$ matrix $\mathbf{W}^*_{intra}$ which gives the allocation to give to each asset within each cluster:

$$\mathbf{W}^*_{intra} = \{\mathbf{w}^{*T}_k | k = 1, \ldots, K\} \tag{3.11}$$

With this, we reduce the filtered covariance matrix to $(K \times K)$ dimensions as a preliminary step to computing inter-cluster weights by performing:

$$\hat{\mathbf{\Sigma}}_{inter} = \mathbf{W}^{*T}_{intra} \hat{\mathbf{\Sigma}} \mathbf{W}^*_{intra} \tag{3.12}$$

Using $\hat{\mathbf{\Sigma}}_{inter}$ we perform mean-variance optimization again to obtain the optimal vector of weights $\mathbf{w}^*_{inter}$ which gives us the allocations to attribute to each cluster. Finally, to get the weight vector $\mathbf{w}^*_{nco}$ that represents the allocation to be given to each asset, we compute:

$$\mathbf{w}^*_{nco} = \mathbf{W}^*_{intra} \mathbf{w}^*_{inter} \tag{3.13}$$

Chapter 4

Empirical Analysis

4.1 Methodology

In this section, we perform an empirical analysis of the theory presented in chapter 2 and chapter 3 in the context of the active management of an equity portfolio. To do this, we start by using supervised machine learning models trained on cross-sectional data composed of technical and fundamental indicators. These models aim to predict which stocks will outperform the equally weighted S&P500 benchmark over the next quarter. In a second step, we use these predictions to build a portfolio of assets that we will rebalance on a quarterly basis. The out-of-sample performance of this scheme is evaluated through a walk-forward simulation ranging from August 2004 to May 2021. We evaluate the machine learning models using standard performance metrics that aim to show the ability of the models to reliably pick the stocks that will outperform their benchmark over the coming quarter. We compare the performance in this respect of 10 machine learning models with a naive method of randomly selecting assets to include in the portfolio. To measure the monetary performance of the portfolios constructed from these predictions, we consider metrics of pure returns, risk and efficiency when transaction costs are included. We compare 4 different portfolio construction techniques that use the predictions of the 10 machine learning-based selection methods plus the baseline. These monetary performance measures are also compared to the same optimization schemes when no stock selection is performed.

4.1.1 Data

Universe

For this analysis, we focus on the stocks included in the S&P500 index. This index includes the 500 largest companies in terms of market capitalization in the United States. We use this index because the stocks that make it up are very liquid. This allows us to more readily assume that our trading activities would not have too much impact on the markets. The index is updated by Standard & Poor's every quarter, where new assets enter the index and those that no longer meet the criteria are removed. We gathered the historical constituents from the Core US Fundamentals Data database available from the Sharadar vendor on the Quandl data platform (Quandl, 2021a). Figure 4.1 shows the evolution of the number of securities included in the index. Given the evolving constituents, we have been cautious at every stage of our analysis to ensure that we have not used data from a stock that was not yet or no longer included in the index at the time of consideration.

Technical Analysis

We downloaded daily price and volume data from the Sharadar Equity Prices database also available on the Quandl data platform (Quandl, 2021b). For computational reasons, we decided to resample these data points from a daily to a weekly frequency. As Wolff and Echterling (2020) do, in performing this process, we considered resampling every Wednesday of the week to avoid potential start and end-of-week effects. We used the last available price as of every Wednesday evening as our weekly price and considered the sum of the volumes between the last Wednesday and the current one for our weekly volume. Based on this weekly pricing and volume data, we have calculated 25 technical indicators inspired by Wolff and Echterling (2020) and Kakushadze (2016). These indicators inform a potential imbalance in supply and demand and can be grouped as momemtum indicators, mean reversion indicators, statistical moments, buying or selling pressure and correlations between returns, volatility and volume. To avoid including redundant indicators, we removed one of the indicators when a pair had an absolute correlation of more than 0.9. Table 4.2 describes each of these 23 remaining technical indicators.

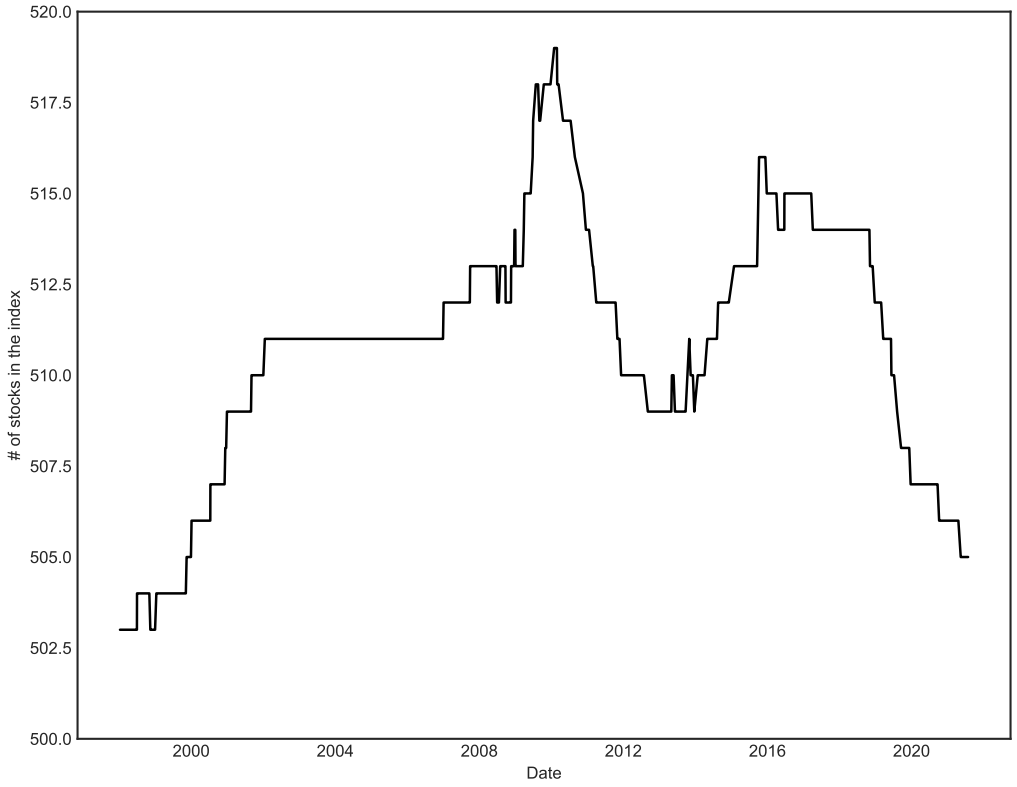

Figure 4.1: *Historical constituents of the S&P500. Since 1998, we find that at any given time there were between 503 and 519 stocks included in the index. The reason why there are more than 500 stocks is because some companies have two common share classes. Hence, it is possible that both types of shares are in the index.*

Name	Description	Period (Weeks)
bb_pr_up	log(Price/Upper Bollinger Band)	13
bb_pr_dn	log(Price/Lower Bollinger Band)	13
rsi	Relative strength index	13
mom	Price momentum	13 & 26
lr_slope	Linear regression slope	13
ma	log(Price/Moving Average)	26 & 52
ret	Lagged return	13& 26
excess	Lagged excess return	13 & 26

ret_div_vol	Risk-adjusted return	13
vol	Return volatility	13
beta	Stock beta	13 & 26
skew	Return skewness	13 & 26
corr_rtn_volume	Correlation between return and volume	13
corr_rtn_spxrtn	Correlation between return and benchmark return	13
obv	On balance volume	-
dlr_vol	Dollar volume	-

Table 4.2: *Description of the technical indicators. Each technical indicator that has a defined period is calculated on a rolling window that goes from the moment of the considered sample to the number of previous weeks. For indicators that do not have a period, they are calculated using the relevant sample values. The technical indicators were calculated using the TA-Lib computation library.*

Fundamental Analysis

As indicated by Cao and You (2020), technical indicators combined with fundamental indicators allow for better predictions of future returns. Therefore, we have sourced Sharadar the Core US Fundamentals Data table available from the Quandl data platform (Quandl, 2021a), quarterly cross-sectional observations of items included in the income statement, balance sheet, cash flow statement as well as standard accounting ratios indicating the financial health of the considered securities. To avoid look-ahead bias, these variables were gathered in a point-in-time fashion which means that for the reporting dates, we made sure that the reported values were in line with what was available at the time from the SEC website. Furthermore, as the time of publishing of each quarterly report varies amongst companies, we could not verify if they were available before the open and chose to shift the reporting day to next trading day. In all, the database contained 194 variables of quarterly observations. These include 94 factors of quarterly observations of quarterly duration and the same 94 factors but with a one year duration. We removed all those that had more than 20% of missing data for the stock universe and

time frame considered. Finally, we removed the variables that had absolute correlations superior to 0.9. As such, we were left with 90 fundamental factors, which are detailed in Table A.2. These factors are for some reported in currency units, others in percentages, ratios and on a per-share basis. Therefore, to ensure the stationarity of our variables, we have calculated the percentage change from one period to the next for the variables denominated in dollars. For the others, we took the first difference from one period to the next. As a means to compare values across different assets, we normalized the per-share variables by multiplying them by the number of outstanding shares and dividing by the enterprise value before performing the first difference. We validated the stationarity of our data with an augmented Dicky-Fuller statistical test (Dickey & Fuller, 1979).

Since the technical and fundamental variables do not have observations at the same frequency, we upsampled the quarterly fundamental data to match the weekly timeframe of the technical indicators. To fill the gaps created by the upsampling procedure, we used the last valid observation. As a result, the fundamental variables remain the same for 13 weeks of observations in the merged dataset. Finally, for each observation in the data matrix that includes the technical indicators and fundamental factors, we removed any observation belonging to a security that was not in the S&P500 index on that date. We thus end up with a data matrix $\mathbf{X}$ of dimension (581118×114). This, however, does not mark the end of the transformations performed on the dataset. Others include imputation, standardization and PCA in order to arrive at the $\tilde{\mathbf{X}}$ data matrix which is used to train the machine learning models. As we do not perform these transformations on all the data at the same time as a means of not creating look ahead bias, we detail these other transformations in subsection 4.1.2 where we present the cross-validation procedure used.

4.1.2 Model Training

To capture the ever-changing nature of the markets, we re-train the stock selection models each year. To do this without introducing look-ahead bias, we conduct a walk-forward simulation by considering at each yearly training date the data available during the preceding 5 years. As we wish to train our stock selection models on data spanning 5 years but do not know which hyperparameters to use, we perform a time series cross-validation of 2 folds following the method exposed in section 1. This part of the process

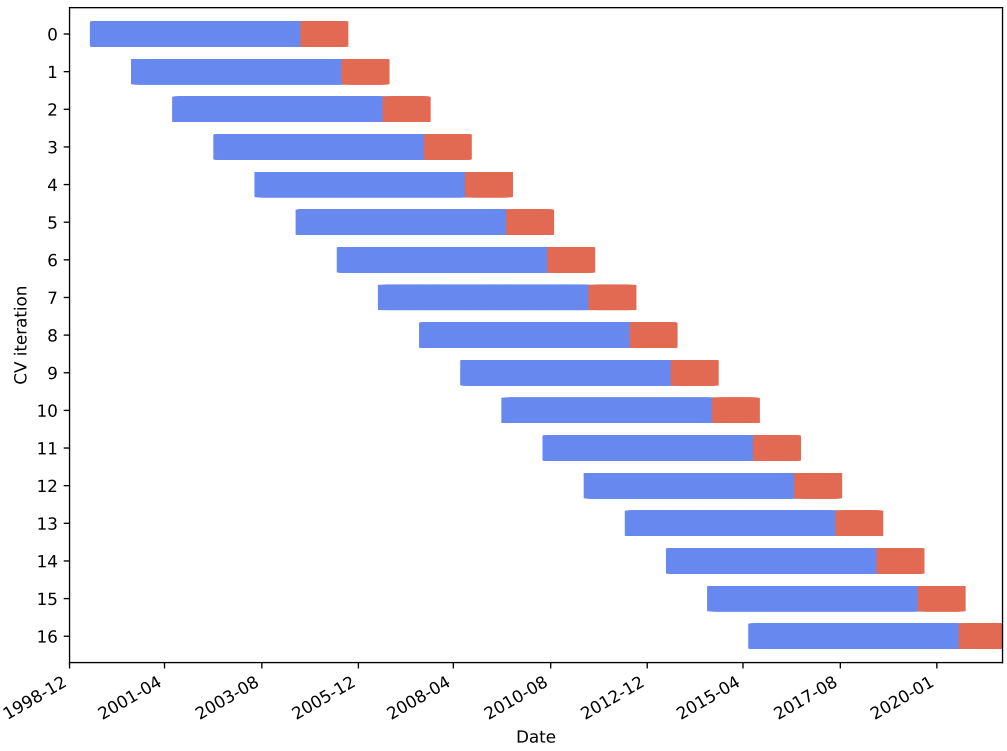

Figure 4.2: *Walk-forward simulation for model training. The training periods are marked in blue; they last for 5 years. In red, we show the out of sample periods, which are used to calculate the performance measures.*

is called the tuning of the hyperparameters. To carry out hyperparameter tuning, we consider for each model a grid that includes all common combinations of hyperparameters. Then, we randomly sample 20 combinations from this grid and perform the time series cross-validation procedure for each sampled hyperparameter combination. For this inner cross-validation loop, we consider a test period of one year and a training period of 3 years as illustrated in Figure 4.3. The selected hyperparameters are the ones that obtain the best average precision (Equation 4.2) during the cross-validation procedure. We then train the model using the best hyperparameters on the 5 years of data. We perform this process for each of our 10 models, every year from 2004 to 2021. This leaves us with a period of 17 years out of sample for which we measure the performance of our stock selection models compared to a random selection. We consider as mod-

els, the logistic regression with regularization lasso (l1), ridge (l2) and elastic-net (en) (log_reg_l1, log_reg_l2, log_reg_en) as defined in chapter 2. To this, we add the linear support vector classifier (lin_svm), the support vector classifier with radial basis function kernel (rbf_svm), the neural network (nn), the decision tree (dt), the random forest (rf), the gradient boosting tree classifier (gbm) and finally, the ensemble model (ens).

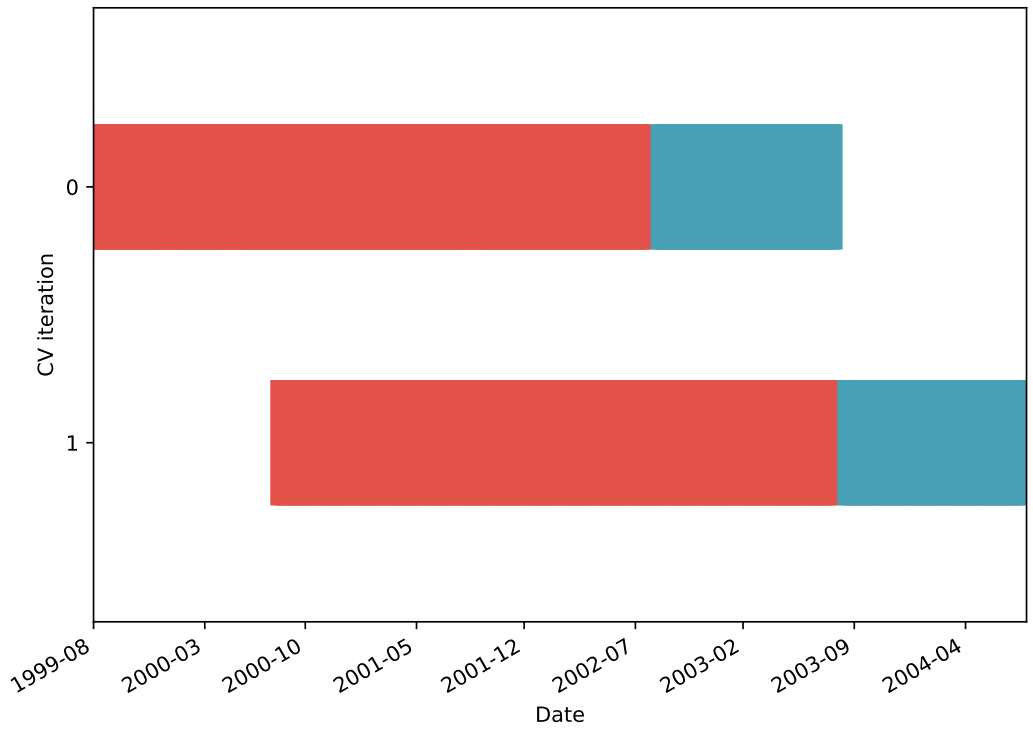

Figure 4.3: *Inner cross-validation loop. We consider here the first inner-cross-validation procedure which represents the blue part of iteration 0 of Figure 4.2. This phase of cross-validation is used to find the hyperparameters of the models which are then trained over the whole 5 years and for which the predictions are used during one year. In this respect, this inner cross-validation consists of two folds, each with a training period of 3 years and a test period of 1 year.*

Here again, we make the necessary steps to prevent the leakage of future data that we would not have had access to at the time of use. Therefore, the last transformations to the data are done through a pipeline just before training a model, whether it is in the hyperparameter tuning procedure or in the training of the model on the 5 years long data. The final transformations include imputing the missing values with the median,

standardizing the input features of the data matrix $\mathbf{X}$ by removing the sample mean $\hat{\boldsymbol{\mu}}$ and dividing by the sample standard deviation $\hat{\boldsymbol{\sigma}}$. Once this is done, we reduce the dimensionality of the data by applying Principal Component Analysis (PCA). The choice of the new dimensionality K is made by applying Equation 2.56 where we choose the dimension that allows us to retain 85% of the variance of the data. This leaves us with the data matrix $\tilde{\mathbf{X}}$ which is used to train the machine learning models.

4.1.3 Portfolio Construction

Once we have trained the stock selection models, we use their predictions for one year. This amounts to 52 buy recommendations per model during the course of the year. However, since we have trained the selection models to predict whether a stock will outperform the benchmark over the next quarter or not, we choose to only act on the predictions each quarter. This also prevents us from incurring excessive trading costs. Therefore, each quarter we observe the holdings of the portfolio from the previous quarter and calculate the vector of weekly returns. Then, for the stocks that are no longer being recommended for investment, we exit the position. For the rest, we form a matrix of weekly returns spanning the preceding 5 years. From this matrix of returns, we estimate the covariance matrix and perform portfolio optimization. We consider 4 different optimization methods: a naive one that assigns an equal weight (ew) to each stock, a weighting scheme calculated by the classic variance minimization (mv) that uses the sample covariance matrix as input, another that uses the covariance matrix estimated after the shrinkage operation (shrink) and finally, an allocation given by the Nested Clustered Optimization (nco) procedure. For each of these methods, we consider non-negativity constraints for the weights. In so doing, we do not permit short-selling. Once we have computed the desired weights, we compare them to the current portfolio allocations and rebalance accordingly. We do this for the 10 selection methods, the random selection method and also for the S&P500 universe where no selection is performed. We include trading costs with each order and use the standard commission structure offered as of August 2021 by the Interactive Brokers brokerage firm. The cost per trade is \$0.005 per share with a minimum of \$1 per trade and maximum of 1% of the trade value. We consider an initial portfolio value of \$10 million as we trade liquid stocks and could therefore readily purchase and dispose of shares without having a large impact on the market.

4.1.4 Performance Evaluation

Classification Metrics

We are initially interested in the ability of machine learning models to select stocks that will outperform their benchmark over the next quarter. In the case of a binary classification, we can evaluate the performances of the classifiers through 4 fundamental quantities:

- *True positive (tp)*: Denotes the number of stocks that were predicted to outperform the benchmark and actually did so;
- *True negative (np)*: Denotes the number of stocks that were predicted to underperform the benchmark and actually did so;
- *False positive (fp)*: Represents the number of stocks that were predicted to outperform the benchmark but did not;
- *False negative (fn)*: Represents the number of stocks that were predicted to underperform the benchmark but did not.

These fundamental quantities are used to give us more information about the quality of our predictions.

- *Accuracy* is the most commonly used performance measure in classification tasks. It represents the proportion of samples that were correctly classified and can be written as:
$$acc = \frac{tp + fp}{tp + np + fp + fn} \tag{4.1}$$
The main concern with this measure is that if the classes are not balanced, the measure is misleading. Indeed, when dealing with an unbalanced dataset, a naive classifier that always predicts the most frequent class would appear to be a very good one, which is not the case. Furthermore, in this study we are particularly interested in the committed error when selecting assets that will outperform their benchmark as we would incur financial losses if this were not the case.

- To know more about the classifier's ability not to recommend under-performing stocks, we use the *precision* defined as:
$$p = \frac{tp}{tp + fp} \tag{4.2}$$

Pushing a model to achieve good precision would, however, be at the expense of the quantity of stocks that are selected and thus carries an opportunity cost.

- The *recall* measures the ability of the classifier to find all the stocks that will outperform the benchmark. It is defined as:

$$r = \frac{tp}{tp + fn} \tag{4.3}$$

- As a compromise, the f_1 *score* measures the performance of the classifier as a harmonic average of precision and recall:

$$f_1 = 2\left(\frac{p \times r}{p + r}\right) \tag{4.4}$$

Backtesting Metrics

Once we have the stock selections made by the models, we are now interested in the economic significance of these when included in a portfolio. As such, we present below back-testing metrics that are commonly used by active fund managers (De Prado, 2018). We can group backtesting metrics into ones that aim to inform us about the performance of a portfolio in terms of returns, risks and efficiency.

Let V_0 be the inital value of the portfolio at the beginning of the backtest, V_T the value of the portfolio at the end of the backtest and $\mathbf{r}_p = \{r_t | t = 1, \ldots, T\}$ the vector of quarterly returns of the portfolio p.

Returns

- The *total return* is defined as:

$$R = \frac{V_T - V_0}{V_0} \tag{4.5}$$

- *Annualized returns* are the returns earned over a period scaled back to a 1-year period. By scaling, we are able to compare the returns of any asset over a period of time. It is expressed as

$$AR = \left(\frac{V_T}{V_0}\right)^{\frac{1}{S}} - 1 \tag{4.6}$$

where T is the number of years and S is the number of periods to form one year.

- The *win rate* measures the fraction of positions that yielded a positive return:

$$WR = \frac{1}{T}\sum_{t=1}^{N} \mathbb{1}_{\{r_t>0\}} \tag{4.7}$$

- The *average gain/loss* for when we generate a profit or endure a loss is given by:

$$\begin{aligned} AG &= \frac{1}{T}\sum_{t=1}^{T} r_t \mathbb{1}_{\{r_t>0\}} \\ AL &= \frac{1}{T}\sum_{t=1}^{T} r_t \mathbb{1}_{\{r_t<0\}} \end{aligned} \tag{4.8}$$

  where $\mathbb{1}\{\cdot\}$ is the indicator function.

Risk

- *Annualized volatility* measures the dispersion of the returns around their mean, expressed as:

$$Vol = \sqrt{\frac{\sum_{t=1}^{T}(r_t - \bar{\mathbf{r}})^2}{T-1} \times 252} \tag{4.9}$$

- *Value at risk (VaR)* is the maximum expected loss of a portfolio due to adverse market movements over a given time period and confidence interval. VaR is thus the quantile of the probability of loss. We denote by L_T the random variable that represents the loss of a portfolio over a period length T; the VaR at confidence interval $1-\alpha$ can be defined as the smallest number $VaR_{1-\alpha}$ (Tardivo, 2002) such that:

$$\mathbb{P}\left\{L_T \leq VaR_{1-\alpha}\right\} \geq 1-\alpha \tag{4.10}$$

- The *Conditional Value at Risk* (CVaR) measures the average loss encountered beyond the VaR cutoff:

$$CVaR_{1-\alpha} = |T|^{-1} \sum_{L_T > VaR_{1-\alpha}} L_T \tag{4.11}$$

  where $|T|^{-1}$ is the proportion of times the random variable L_T has values beyond the VaR threshold.

Efficiency

- *The Sharpe ratio (SR)* measures the average returns obtained in excess of a risk-free asset per unit of volatility. Let $\hat{\mu}$ be the sample mean of the return of the strategy in excess of a risk-free asset and $\hat{\sigma}$ its sample standard deviation, the SR can be estimated as:

$$\hat{SR} = \frac{\hat{\mu}}{\hat{\sigma}} \tag{4.12}$$

4.2 Results

4.2.1 Classification

Once the all the models are trained and tuned in the walk forward simulation outlined in subsection 4.1.2, we calculate the classification performance measures on the aggregation of all out-of-sample periods. Regarding the ability of our scheme to select stocks that will obtain better returns than the benchmark, we see that the overall performance is quite encouraging (Figure 4.4).

As far as accuracy is concerned, Figure 4.5 shows that for the most part, the classes are balanced over the evaluation period. However, the ratio of outperforming assets evolves over time. Therefore, as a baseline approach, we consider stratified random predictions. That is, the distribution of the classes observed in the training set is maintained during the out-of-sample period. With this in mind, the baseline model scores an accuracy of 49.9% which is in line with what we would expect. All but one of the machine learning models obtain a better accuracy than this. The nn achieves the best accuracy with 50.8% of correctly classified samples ahead of the log_reg_l1 and the ensemble model. The decision tree obtains the lowest accuracy of all the models, being on par with the baseline method. Theses accuracy scores may seem low when compared to other disciplines, where cutting-edge models obtain scores of more than 90% (Zhai et al., 2021). This is however very rarely the case in finance due to the low signal to noise ratio. Nevertheless Grinold and Kahn (1999) show that it is not mandatory to have a very high accuracy in finance because we are mainly interested in the economic impact of our predictions. The latter can be very significant even when the accuracy is relatively low. Suppose a simple case where an investor either wins or loses \$1 on each trade. If their accuracy is 50%, then their

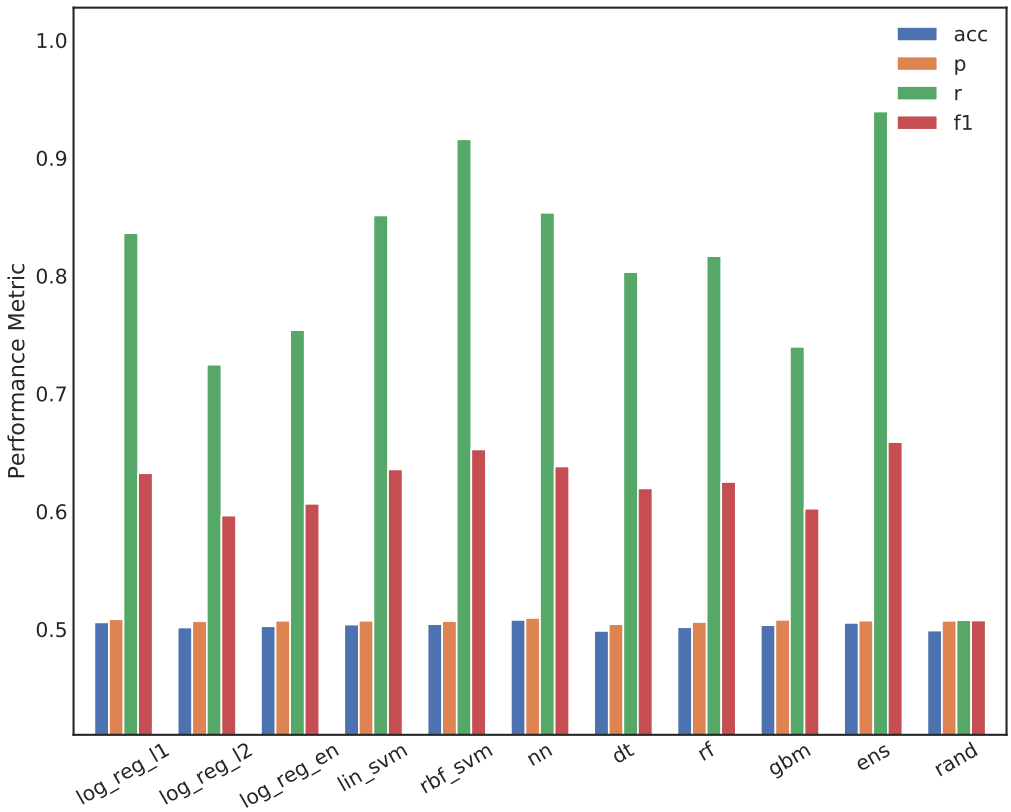

Figure 4.4: *Classification metrics. This figure displays the same information as Table A.4. However, it allows us to see more clearly the differences between the models. The accuracy and precision are very similar accross the approaches. However, in finance this is commonly the case and a small difference is accuracy and precision can make a big difference to the equity of an investor. The random selection is the one that performs the worst of all the approaches. In terms of recall, the ensemble model obtains the best performance, followed closely by the rbf_svm. The ranking is maintained for the f1 score. The machine learning models predict that many of the samples will be outperformers. They therefore have a high generality but a low pericision.*

expected profit is \$0. However, if the investor's accuracy is 51%, the expected profit rises to \$0.02 per trade. Now let's imagine that when the investor wins, they earn \$1.1 and still lose \$1, the expected payout for each trade is now \$0.07. If this pattern is repeated every week over 20 years, one quickly realizes the economic impact that a seemingly slight positive expectancy can have in the markets. This is all the more significant if one considers that the gains are compounded as shown is Figure 4.6.

As far as precision is concerned, we find here that the results are very similar between

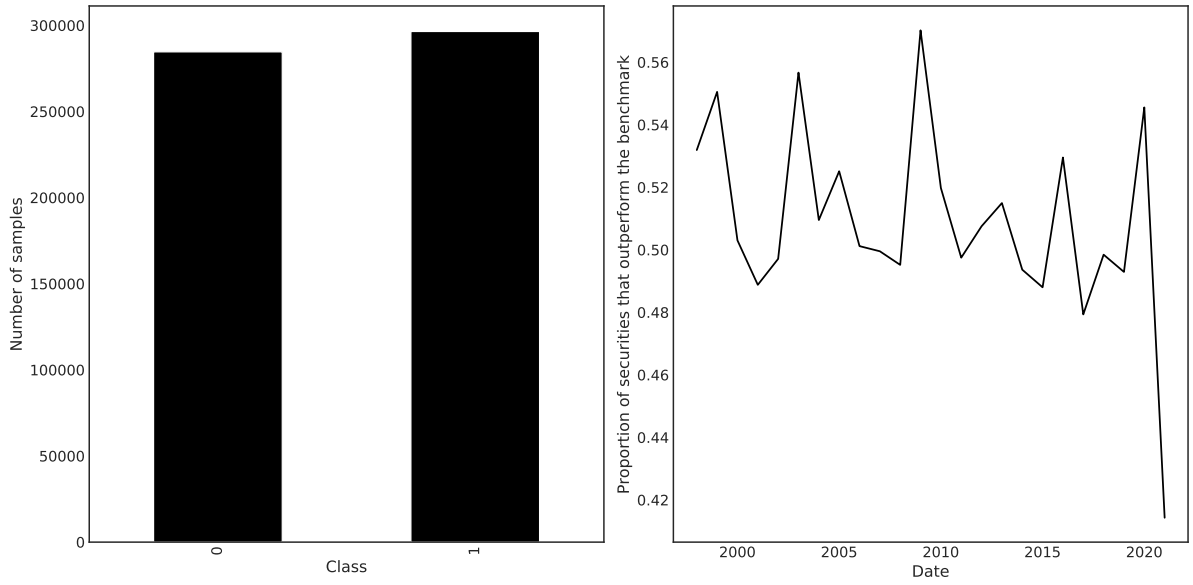

Figure 4.5: *Distribution of the classes. The classes are evenly distributed across the entire sample period (left figure). However, when we look at the distribution of classes through time (figure on the right), we see that it is fluctuating. It is interesting to note the drop in 2020 which is probably due to the aftermath of coronavirus crisis where many companies that were more strongly affected by the crisis are struggling to outperform the index. As a reminder, they are encoded as follows: If the asset outperforms the benchmark over the coming quarter, this is denoted by a 1, the opposite is encoded by a 0.*

all approaches. The neural network obtains again the highest score while the decision tree has the lowest score one more time. In general, the ability of the classifiers not to label an asset as an outperformer when it is not is weak. Here, machine learning models obtain results that are comparable to the random baseline model. This suggests that the classifiers have a tendency of recommending stocks in which we do not want to invest.

Looking at the recall however, we see that this lack of performance in precision is compensated by the impressive true positive rate. Indeed, each selection model obtains a much stronger score than the baseline random approach. This indicates that the ability of the classifier to find all the assets to invest in is particularly good. Here, the ensemble model gets the best performance, followed by the nn and the rbf_svm obtaining recall scores of 94%, 91.6% and 85.4%, respectively. The random baseline approach obtains a recall of 50.8%; proving the superior breadth of the recommendations made by the classifiers. The f-1 score, being a harmonic mean of precision and recall, follows the same ranking as for the recall. Indeed, since the differences across all methods are very small

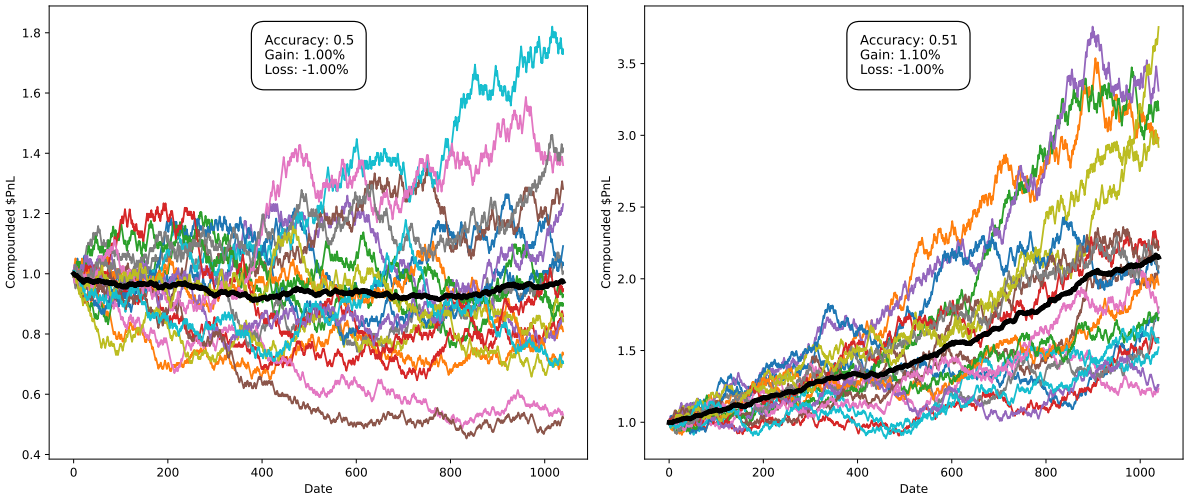

Figure 4.6: *Hypothetical equity curves when there is a positive trade expectancy. The graph illustrates a hypothetical case where, on the right-hand side, an investor has 50% accuracy, a 1% return when a trade is successful and a -1% return when a trade is unsuccessful. We see that if we consider a sample size of 20 52-week periods (20 years), the expected terminal wealth (black line) is zero. On the left side, we see what happens if the accuracy is now 51%, each gain is 1.1% and the losses are still -1%. In this case, the expected terminal wealth becomes very interesting.*

for one and significant for the other, the latter takes the most weight (Figure 4.4).

These results allow us to verify empirically that in the S&P500 universe, machine learning techniques are able to offer informative investment recommendations when it comes to predicting a great breadth of stocks that will outperform the benchmark in the coming quarter. It is however important to consider the economic impact of these recommendations, which is what we do next.

4.2.2 Backtesting

In this section, we measure the economic significance of the predictions made by the aforementioned models. The returns of the portfolio are computed weekly and we report the performance in percentages. In addition, we compare the returns of the selection models to the approach where all stocks in the index are considered. The performance in terms of returns, risks and efficiency is shown in Table A.6.

We first note that the weekly returns are strongly correlated (as seen in Figure 4.7) for all the methodologies considered. This is to be expected since we are considering

long-only portfolios that are part of a universe with a large market mode (MacMahon & Garlaschelli, 2013). Moreover, we find that the solutions produced by the **mv** and **shrink** optimization methods are very similar. This is most likely caused by the fact that in imposing non-negativity constraints, we perform a regularization similar to the shrinkage operation on the covariance matrix (Ma & Jagannathan, 2003). Moreover, we notice that the **nco** approach is very similar to the **ew** method which assigns equal weights to each asset in the portfolio.

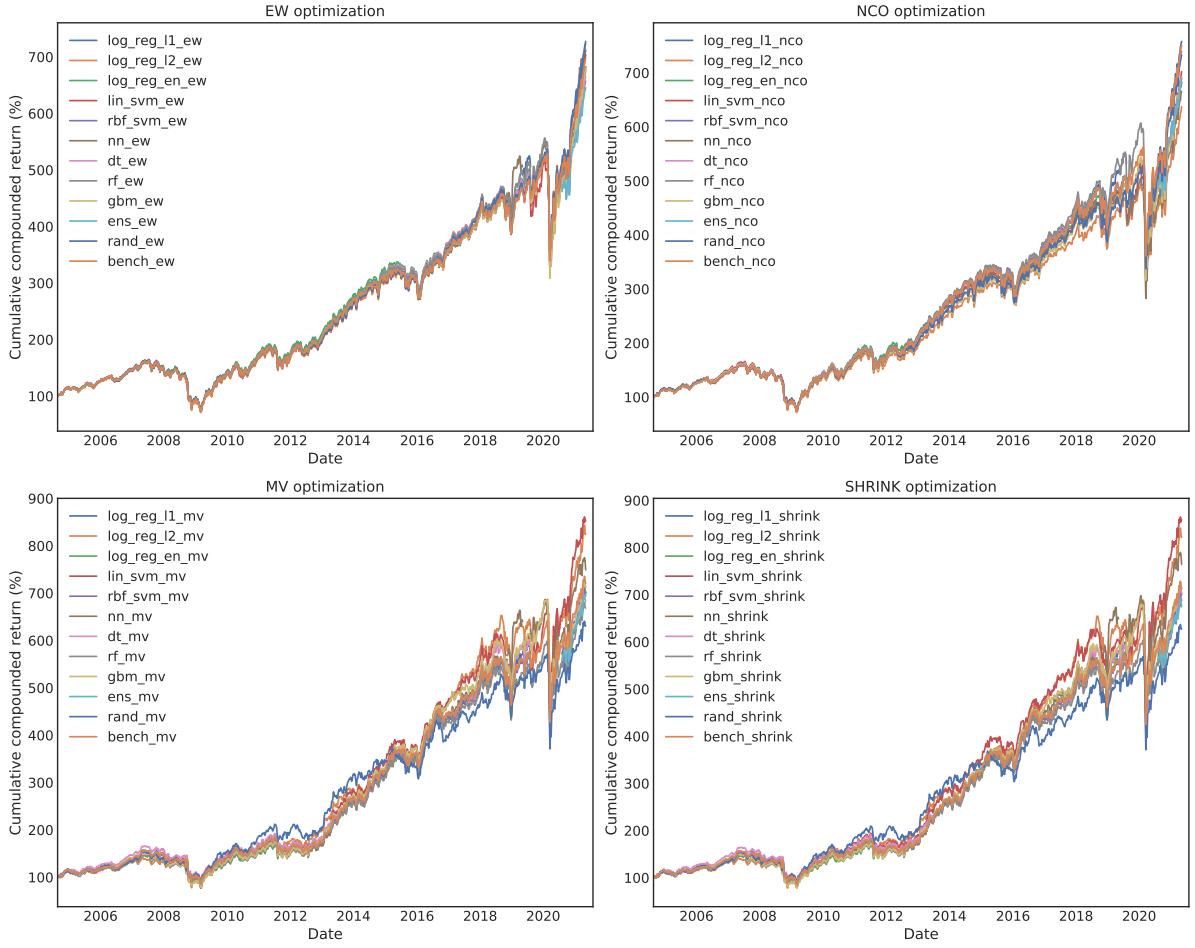

Figure 4.7: *Cumulative returns of the different approaches. It can be seen that the **ew** and **nco** approaches are very similar (top 2 figures). So are the **mv** and **shrink** (bottom 2 figures). All the portfolios optimized with the **ew** (top right figure) are very similar and obtain performances that are almost identical. For the **nco** (top left figure), while it is similar to the **ew**, we still see that various selection models react differently to the suggested weights. This variation is even more pronounced for portfolios constructed with **mv** and **shrink**. For some of them, they benefit strongly from these methods and obtain higher returns than the best portfolio built by **ew** or **nco**.*

From Figure 4.8, we see that in terms of pure returns, the **lin_svm** obtains the best performance with a total return of 759.2% followed by the **log_reg_l2** which obtains a total return of 754.5%. These models find their best results when the **mv** and **shrink** optimization algorithms are used. Moreover, when optimized with these two methods, the random selection performs the poorest with a total return of 527.16 %. This is however not the case when optimizing with **ew** and **nco**, where the random selection is comparable with most of the machine learning algorithms and the non-selecting method. The same conclusions can be drawn by analyzing the cagr in Figure B.1.

As far as the win rate is concerned, we see from Figure 4.9 that **log_reg_l1** is the best performing with a 61.51% hit ratio. It is closely followed by the the **nn**; these two were the most precise when assessing the quality of the classifications. For all selection methods, the pair **ew** and **nco** have a better win ratio than the other two. Even for this performance measure, random selection is not the worst performing and for the most part stands up to the machine learning methods. This is explained in large part because the selection is made on an index that has been on a rising trend throughout history.

When, considering the average gain per trade, the **gbm** is the best performing model with the **dt** and **rf** following closely behind. They have an average positive return of 2.187%, 2.182% and 2.180%, respectively. Figure 4.10 shows that the tree-based selection models present a tendency of selecting assets that yield the highest returns. We also note that the models we presented as having the highest winning ratio are also the ones that have the lowest average gain per trade. As no single selection method appears to dominates the others in all return performance measures for the considered optimization methodologies, it is expected that a model which selects assets that yield large average positive returns will participate in more trades that yield a negative return. It appears that the portfolios constructed by **ew** and **nco** enjoy the highest average winning trades for all the considered selection methods.

In summary, for each pure return performance measure, there are several machine learning-based selection models that outperform both random and no selection approaches. This is however not the case for all machine learning models. They thus seem to each have their strengths and weaknesses depending on the measure considered. As such, it is surprising to see that the ensemble model is not the one with the best overall performance. While such findings have been observed in other applications (Atiya, 2019),in the

case of stock selection, the combination of models allows for greater generality (in terms of the proportion of outperforming assets that are recommended) rather than precision. While this offers lower idiosyncratic risk due to greater diversification, in terms of pure returns, the combination of models does not recommend the best performing portfolios.

In terms of risk, we see from the Figure 4.11 that the **log_reg_l2** model is the only selection model whose average loss is of the same order as the average gain. All other selection methods have average losses of about -2.5%, which is greater than the average gain. Figure 4.12 illustrates that there seems to be a cluster of models that obtain a volatility that is on average 15% lower than the others. This cluster includes the **log_reg_l1**, **log_reg_en**, **rbf_svm**, **dt** and **ens** selection schemes. Among these, we find most of the models that obtained the best recall scores. These selection models are thus the ones that have the best coverage by selecting the largest number of assets that outperform the benchmark each quarter. As such the resulting portfolios contain less idiosyncratic risk and are more diversified. The **shrink** and **mv** are the allocation methods that obtain the smallest out of sample portfolio variance, for all selection schemes. It is an encouraging result to see that portfolio optimization performs as intended when we restrict the values taken by the weights and shrink the covariance matrix. The decomposition performed in the **nco** on the other hand seems to be such that the allocation retained is similar to the equal weight portfolio. These findings are the same when we consider the var and cvar measures displayed in Figure B.2 and Figure B.3, respectively.

In terms of efficiency, Figure 4.13 reveals that the ratio between return and risk is best for the **log_reg_l1**, it is closely followed by the **lin_svm** and **dt**. These 3 selection methods has Sharpe ratios of 0.811, 8.809 and 0.805, respectively. The latter perform best when the portfolio's are constructed with **shrink** and **mv**. It is interesting to note that when formed with the aforementioned optimization methods, each of the portfolios created from selections made by machine learning models obtain a Sharpe ratio that is about 18% higher than the random selection. Furthermore, 7 of the 10 machine learning-based selection models are more efficient than portfolios containing all stocks in the benchmark. If we only consider the **log_reg_l1**, which is the approach that forms the most efficient portfolios and the portfolio composed of randomly selected assets, we find that the machine model gets a yearly Sharpe ratio that is for the majority of years superior to the randomly composed portfolio (see Figure 4.14).

We can further decompose the portfolio returns to measure the systematic volatility of the proposed portfolios when compared to the benchmark one (Fama & French, 2004). We define here the benchmark portfolio as the equally weighted S&P Index (bench_ew). The decomposition can be written as follows:

$$\mathbf{r}_p = \alpha + \beta \mathbf{r}_b \tag{4.13}$$

where $\mathbf{r}_p$ is the vector of returns of the considered portfolio and $\mathbf{r}_b$ is the vector of returns of the benchmark portfolio. Thus, β is a scalar value which informs us about how the value of portfolio changes when the benchmark fluctuates. The scalar term α represents the return obtained in excess of the benchmark after it has been adjusted for the volatility of the overall market. Regarding β, Figure 4.15 shows that as we had previously noted for the volatility, there are 5 models that show a systematic volatility that is below the others. This implies that for these portfolios, we should expect to see returns around 30% less volatile than the benchmark when the portfolios are optimized using either the **shrink** or **mv** schemes. The same models that outperform in terms of β obtain an excess return of about 4.5% which can not be attributed to a general market movement.

In summary, we have validated empirically that the selections made by the machine learning models allow us to build portfolios that have a better return to risk ratio than when we form portfolios with a random or no selection. Moreover, when constraints are considered and the covariance matrix is adequately pre-processed, we obtain more efficient portfolios than a naive allocation that gives equal weights to each security.

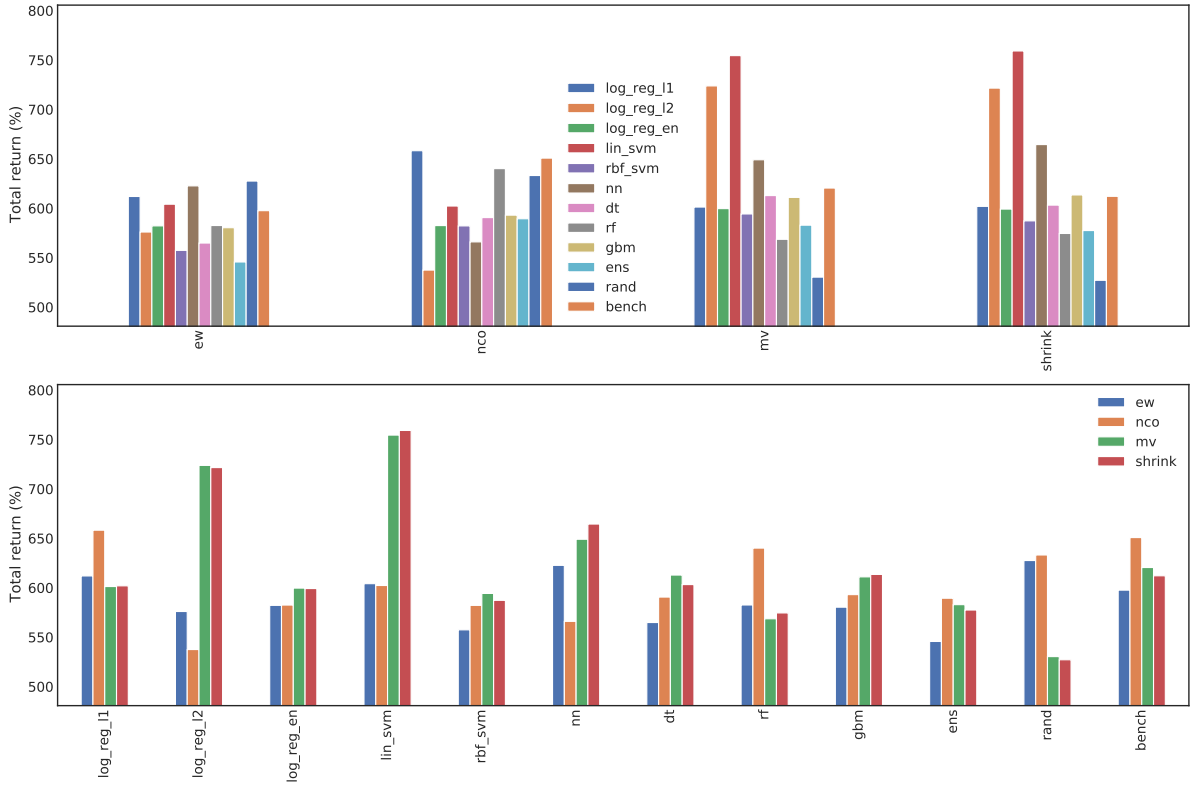

Figure 4.8: *Comparison of the total return. In terms of optimization schemes, different models benefit from different optimization methods. For example,* ***log_reg_l1*** *performs better in terms of returns when constructed with* ***nco*** *than the rest of the methods. Optimization with* ***shrink*** *gets the best performance when selections are made with* ***lin_svm****. The models,* ***log_reg_l1****,* ***log_reg_l2*** *and* ***lin_svm*** *obtain the best performances. It is interesting to see that for this evaluation metric the linear models obtain the best performance. This is not often observed in other financial applications (Choi et al., 2020). This underperformance can be explained by potential overfitting given the flexibility of some non-linear models.*

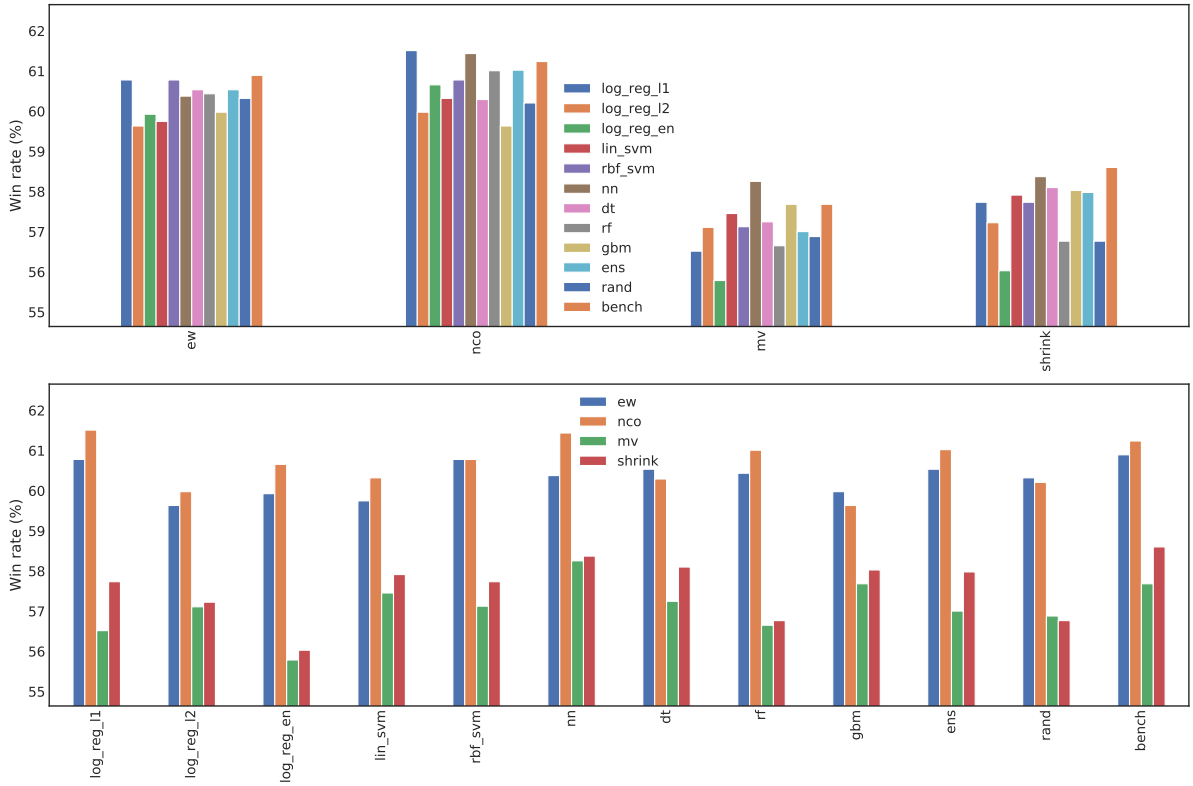

Figure 4.9: *Comparison of the win rate. We observe here that the* ***nco*** *and* ***ew*** *optimization methods generate portfolios that have a higher proportion of winning trades than the* ***mv*** *and* ***shrink*** *methods. We observe here that the* ***nco*** *and* ***ew*** *optimization methods generate portfolios that have a higher proportion of winning trades than the other two methods. The* ***log_reg_l1*** *and* ***nn*** *obtain the best scores in this respect. They were also the ones with the best accuracy.*

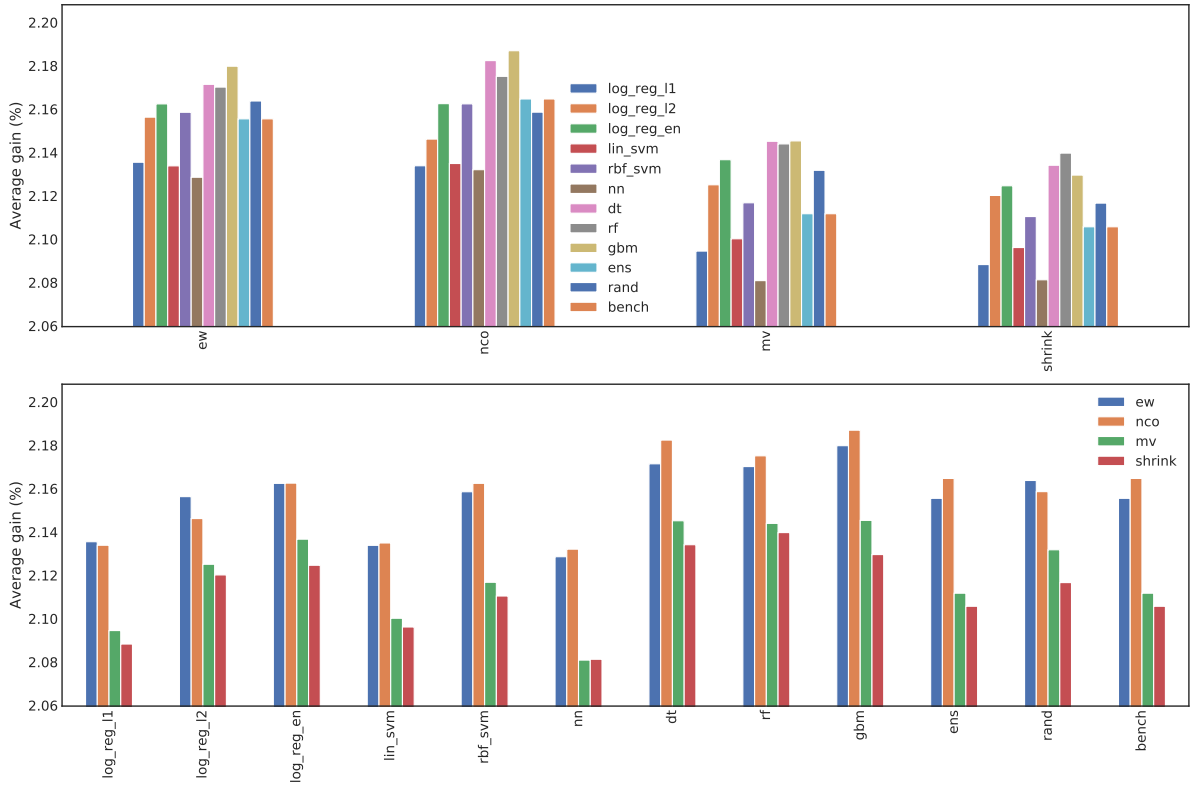

Figure 4.10: *Comparison of the average winning trade. The* ***ew*** *and* ***nco*** *optimization methods obtain for the most part average gains that are higher than the other two optimization methods. We also notice that models based on decision trees, namely* ***dt****,* ***rf*** *and* ***gbm*** *obtain the highest average gains.*

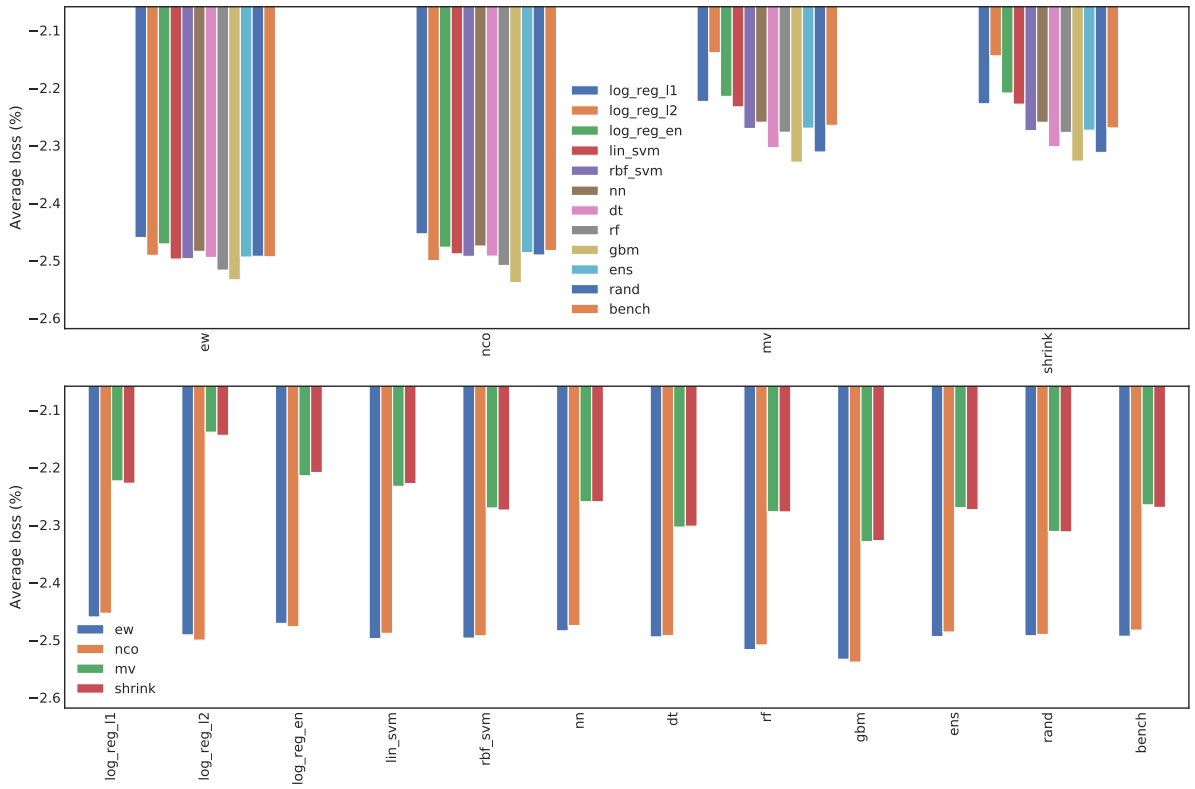

Figure 4.11: *Comparison of the average losing trade. The* ***mv*** *and* ***shrink*** *optimization methods have the lowest average losses. The* ***log_reg_l2*** *has the lowest average loss. In line with the results observed for the average gain, the models based on decision trees have the highest average losses.*

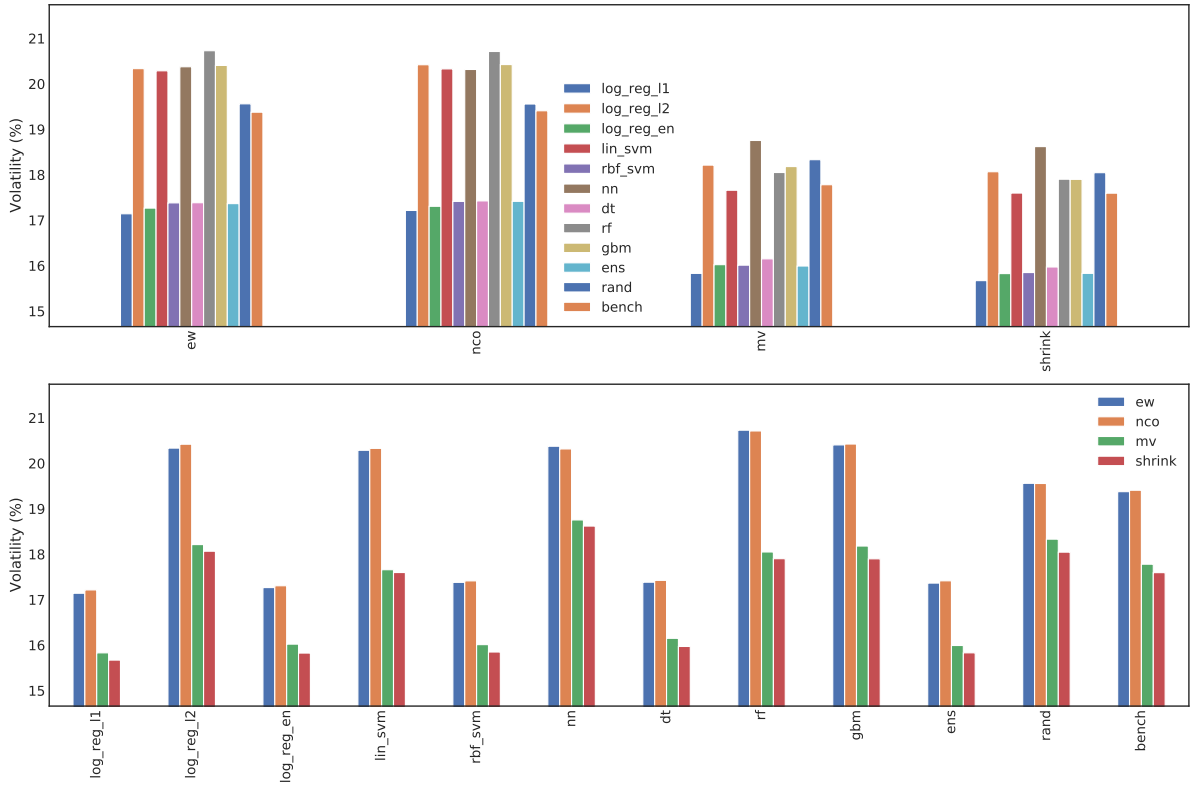

Figure 4.12: *Comparison of the annualized volatility. The* ***mv*** *and* ***shrink*** *optimization methods produce portfolios that have smaller volatility than the other two construction methods. We notice that there are 5 models that have a volatility that is significantly lower than the others. These same models were the ones that enjoyed the greatest recall.*

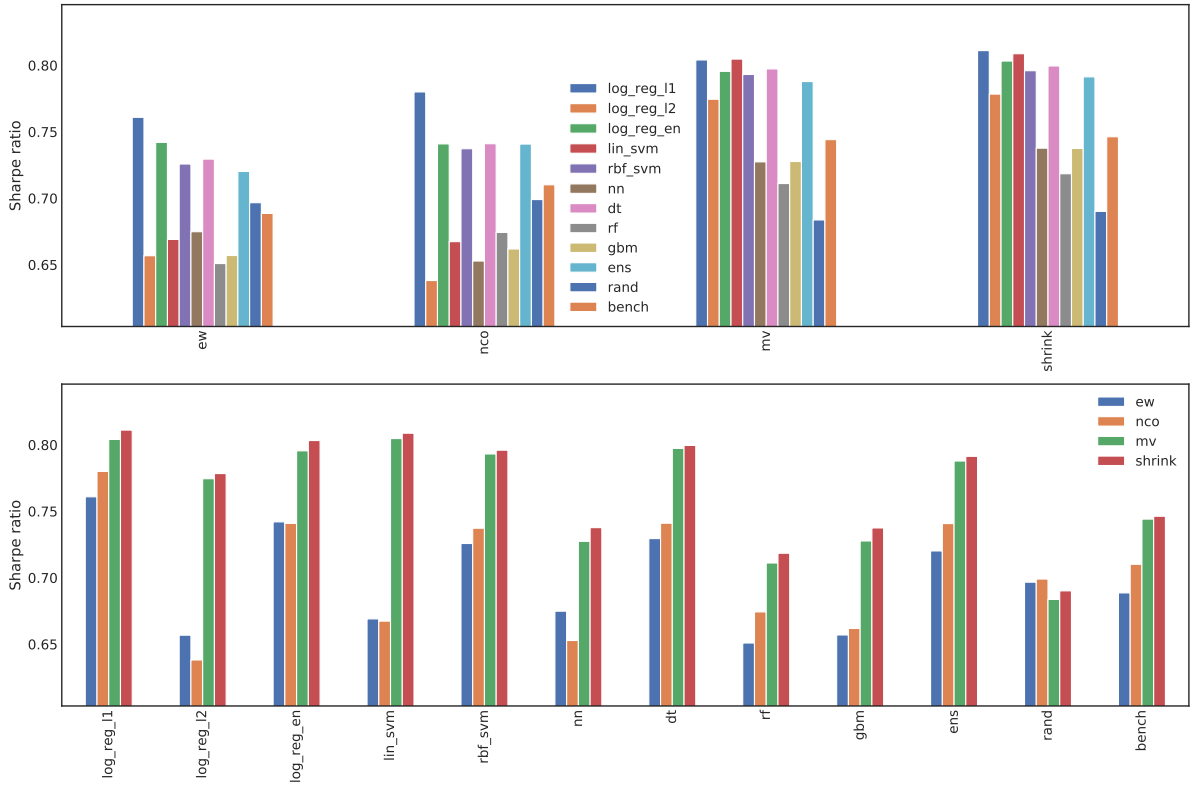

Figure 4.13: *Comparison of the annualized Sharpe ratio. The **mv** and **shrink** optimization methods produce portfolios that have better Sharpe ratios than the other two construction methods. When optimized with **mv** or **shrink**, all selection models achieve better efficiency than random selection. Moreover, the vast majority perform better in this respect than portfolios composed of the S&P500 index.*

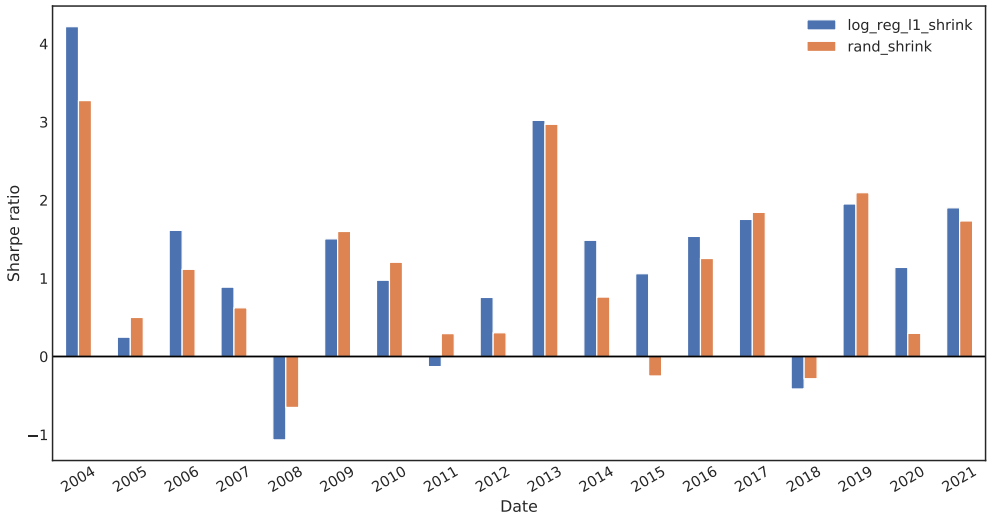

Figure 4.14: *Comparison of the annualized Sharpe ratio computed yearly. This figure compares the Sharpe ratio of the portfolio constructed on the basis of the recommendations made by the most efficient model to the random selection model; each year. We notice that during periods of economic downturn when the portfolios experiences a negative return, the portfolio based on* ***log_reg_l1*** *predictions and* ***shrink*** *optimization is less efficient than the portfolio composed of randomly selected assets also optimized with* ***shrink****.*

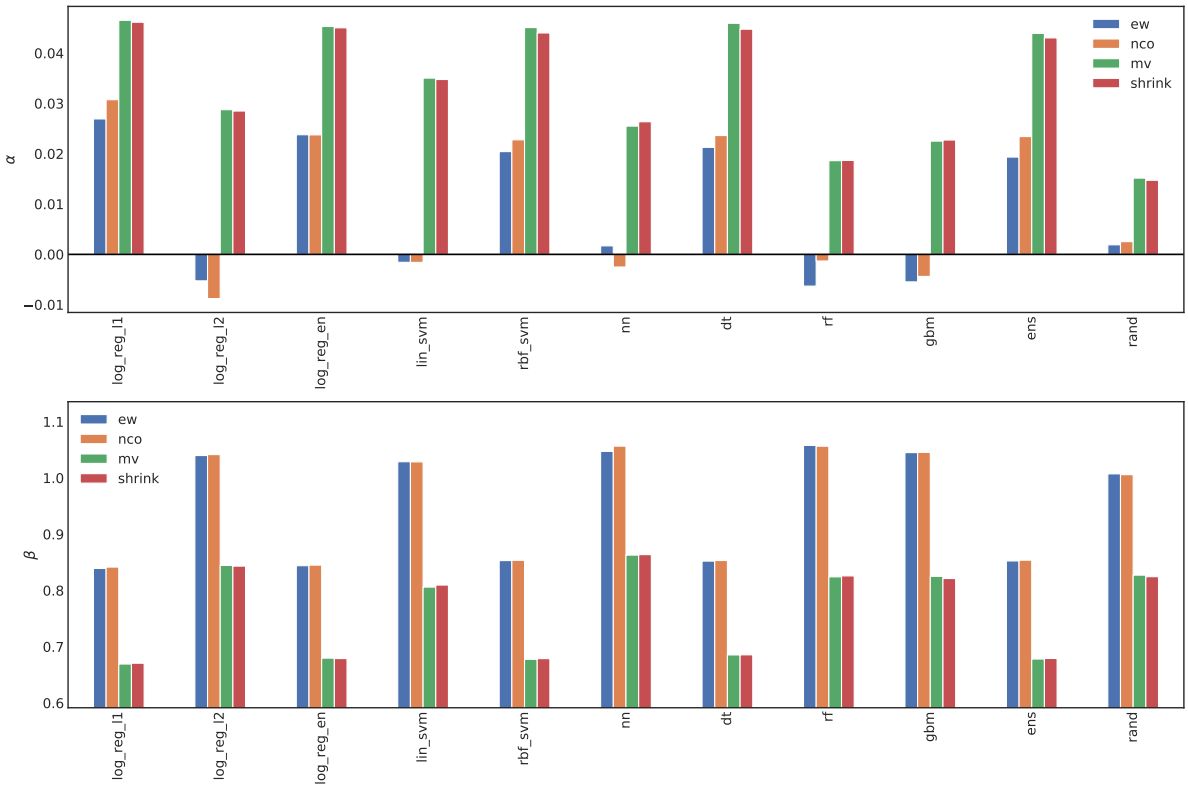

Figure 4.15: *Greeks of the portfolios. The* ***mv*** *and* ***shrink*** *optimization methods produce portfolios that have a positive return in excess of the reference portfolio. We note that all machine learning based selection models perform better than random selection. Regarding the systematic volatility of the portfolios, we see that the models with the highest* α *are also those with the lowest* β*. These models are therefore less volatile than the reference portfolio. This demonstrates the superiority of machine learning models in the selection of financial assets to include in a portfolio.*

Chapter 5

Conclusion

The aim of this thesis was to assess the relevance of machine learning methods in the context of the active management of an equity portfolio. To this end, we considered in a first step the selection of assets that are predicted to outperform their benchmark during the next quarter. For this purpose, we have presented 10 supervised and 2 unsupervised machine learning techniques. These were presented in chapter 2. The first unsupervised algorithm aims at reducing the dimensionality of the data while the second one aims at clustering assets into groups that are anti-correlated between them and correlated among the group. In order to measure the economic importance of the selections made by the machine learning models, we have considered different methods of portfolio construction. Each one aims at correcting the undesirable effects of instability and concentration of the procedure introduced by Henry Markowitz. As such, we presented the classical mean-variance portfolio optimization technique, the shrinkage method used on the sample covariance matrix and the Nested Clustered Algorithm introduced by De Prado (2018). The models were trained on weekly data that includes technical indicators that give an indication of supply and demand forces as well as fundamental factors that give an indication of the financial health of the securities. The universe of assets considered is based on the historical constituents of the S&P500. The models were trained by considering a sliding window of in and out sample periods. To this end, we trained a new model each year using data from the previous 5 years with the objective of making predictions with models that are adapted to relatively recent information and therefore, more informative given the changing nature of the markets. We then evaluated the performance of our approach over an aggregated out of sample period ranging from August

2004 to May 2021. We first evaluated the performance of our models using standard classification measures. To this end, we found that in terms of precision, only some models were better than a random selection of assets. On the other hand, in terms of recall, all machine learning-based models performed significantly better than the baseline. In a second step, we evaluated the performance in terms of returns, risks and efficiency of portfolios built from predictions made by the machine learning models. In this respect, for pure performance measures, only some of the selection models were better than the random selection or the benchmark index. In terms of risk, we found that there was a group of models which selected a larger portion of the stocks that outperformed the index, and thus proposed the construction of more diversified portfolios. These had annualized volatilities that were 15% lower than the other techniques considered. Furthermore, most of our proposed methods were more efficient than the baseline in terms of returns/risks. The top ranked selection model achieved a Sharpe ratio of 0.811 against 0.684 for the random selections.

The completion of this thesis has raised a number of limitations that are of great importance with regard to the results obtained, and which could be the subject of future research:

- We have not analyzed the importances of the features we have included in our models. However, Man and Chan (2020) have shown that the selection and retention of the significant features allows to obtain much better performances out of sample. Furthermore, by understanding what features are the most important in predicting which stocks are susceptible to outperforming their benchmark over the coming quarter, we would be able to discover economic theories which could give confidence in the predictions. We would also be able to derive more features that could add more predictive power to the classifiers;

- As we performed a rolling window simulation and re-trained our models every year since 2004, we assumed that we would have had access to the technology we have today. However, in order to perform this research, a laptop with a 4 core Intel Core i5 processor and 8GB of RAM was unable to perform the analysis without crashing. We were therefore required to use a higher performance cloud based computer on the PaperSpace platform with 30GB of ram and a 12-core processor. These types

of computers were rare and very expensive to access during some of the periods we considered in the analysis. In addition, we used, for example, the random forest algorithm as an asset selection method. However, this algorithm was only proposed in 2006, a period that exceeds the beginning of our analysis. The results may therefore be unrealistic for measures that are further away in time because we have analysed past data using current technology;

- Through this walk-forward simulation, we have in fact only tested one scenario; the historical one. Our results are thus biased by the sequence in which the data points were observed. Therefore, this scenario is in no way a guarantee that the performance of the proposed methods will continue in the same way in the future. In this respect, (De Prado, 2018) presents the combinatorial purged cross-validation method which generates a predefined number of different paths; these allow us to evaluate in a more robust way the backtesting performances. In addition, there are machine learning techniques such as Generative Adversarial Networks, which are neural networks that are trained to generate new, unobserved data samples. It would be interesting as future research to see whether this method would allow us to evaluate several scenarios that would have been generated randomly but still involving the same generating process behind financial data. Thus, enabling more reliable performance measures;

- While we have taken into account direct transaction costs as charged by a brokerage firm, we have not taken into account any transaction costs related to slippage. Indeed, we did not collect any quote data indicating the bid and ask, so we had no indication of the spread. However, for each of our backtests, we assumed that we were able to execute the positions at the closing price. This price is however merely the last price executed on any one of the exchanges where the stock is listed. Therefore, a very small transaction could have been executed on a secondary exchange and we falsely assumed that we would have been able to execute our orders at the same price. This assumption potentially makes our results look advantageous. In fact, if we wanted to get out of our positions at the market close, we would have had to use a market order just before the close. This would then execute at the best bid price if we were looking to sell or at the best ask price if we were looking to

buy. These, could potentially be far away from the price listed as the closing price. Throughout the analysis, we thus assumed that we have bought or sold the assets at a price that is higher or lower than the price at which we could have executed in practice. This methodological limitation therefore impacts the returns, risks and efficiency that an asset manager would have observed if they had actually executed this strategy. It would then be judicious to re-run the analysis while assuming that purchases are made on the ask side and that sales are made on the bid side. Furthermore, by using tick data, it would be possible to approximate the capacity of the strategy; that is, the total capital it could support before negatively impacting the results due to the large orders submitted at once;

- Machine learning is a rapidly evolving field and we have only considered a very small portion of the existing methods; some of which may be more appropriate to the problem at hand. As such, it may be worthwhile to investigate other selection models that are more complex. For example, recurrent neural networks and more precisely long short term memory (LSTM) networks have been shown to be particularly efficient in financial applications that involve time series data (Chen et al., 2015). In reducing the dimensionality, we have used PCA which performs an orthogonal linear transformation. It could therefore be interesting to study nonlinear techniques such as kernel PCA (Chang & Wu, 2015) or an autoencoder. The latter is a neural network trained specifically to reduce dimensionality and has been shown to provide compelling performances in financial applications (Gu et al., 2021).

Bibliography

Amit, Y., & Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural computation*, *9*(7), 1545–1588.

Atiya, A. (2019). Why does forecast combination work so well? *International Journal of Forecasting*, *36*. https://doi.org/10.1016/j.ijforecast.2019.03.010

Black, F., & Litterman, R. (1992). Global portfolio optimization. *Financial analysts journal*, *48*(5), 28–43.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, *2008*(10), P10008.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Cao, K., & You, H. (2020). *Fundamental analysis via machine learning* (tech. rep.). Working Paper.

Chang, P.-C., & Wu, J.-L. (2015). A critical feature extraction by kernel pca in stock trading model. *Soft Computing*, *19*(5), 1393–1408.

Chen, K., Zhou, Y., & Dai, F. (2015). A lstm-based method for stock returns prediction: A case study of china stock market. *2015 IEEE International Conference on Big Data (Big Data)*, 2823–2824. https://doi.org/10.1109/BigData.2015.7364089

Choi, D., Jiang, W., & Zhang, C. (2020). Alpha go everywhere: Machine learning and international stock returns. *Available at SSRN 3489679*.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

De Prado, M. L. (2016). A robust estimator of the efficient frontier. *Available at SSRN 3469961*.

De Prado, M. L. (2018). *Advances in financial machine learning*. John Wiley & Sons.

De Prado, M. L. (2020). *Machine learning for asset managers*. Cambridge University Press.

Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). *Mathematics for machine learning*. Cambridge University Press.

DeMiguel, V., Garlappi, L., & Uppal, R. (2009). Optimal versus naive diversification: How inefficient is the 1/n portfolio strategy? *The review of Financial studies*, *22*(5), 1915–1953.

Denuit, M., Hainaut, D., & Trufin, J. (2020). *Effective statistical learning methods for actuaries ii: Tree-based methods and extensions*. Springer Nature.

Diamond, S., & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, *17*(83), 1–5.

Dickey, D., & Fuller, W. (1979). Distribution of the estimators for autoregressive time series with a unit root. *JASA. Journal of the American Statistical Association*, *74*. https://doi.org/10.2307/2286348

Eckart, C., & Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, *1*(3), 211–218.

Fama, E. F. (1965). The behavior of stock-market prices. *The journal of Business*, *38*(1), 34–105.

Fama, E. F., & French, K. R. (2004). The capital asset pricing model: Theory and evidence. *Journal of economic perspectives*, *18*(3), 25–46.

Friedman, J., Hastie, T., Tibshirani, R., et al. (2001). *The elements of statistical learning* (Vol. 1). Springer series in statistics New York.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of statistics*, 1189–1232.

Götze, F., & Tikhomirov, A. (2004). Rate of convergence in probability to the Marchenko-Pastur law. *Bernoulli*, *10*(3), 503–548. https://doi.org/10.3150/bj/1089206408

Grinold, R., & Kahn, R. (1999). *Active portfolio management: A quantitative approach for producing superior returns and selecting superior returns and controlling risk*. Mcgraw-hill.

Gu, S., Kelly, B., & Xiu, D. (2021). Autoencoder asset pricing models. *Journal of Econometrics*, *222*(1), 429–450.

Herold, U., & Maurer, R. (2006). Portfolio choice and estimation risk: A comparison of bayesian to heuristic approaches. *Astin Bulletin*, *36*. https://doi.org/10.2143/AST.36.1.2014147

Israel, R., Kelly, B. T., & Moskowitz, T. J. (2020). Can machines' learn'finance? *Available at SSRN 3624052*.

Jaggi, M., & Flammarion, N. (2020). Lecture notes: Machine Learning Course - CS433.

Jiang, H., He, P., Chen, W., Liu, X., Gao, J., & Zhao, T. (2019). Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*.

Jobson, J. D., & Korkie, B. (1980). Estimation for markowitz efficient portfolios. *Journal of the American Statistical Association*, *75*(371), 544–554.

Kakushadze, Z. (2016). 101 formulaic alphas. *Wilmott*, *2016*(84), 72–81.

Kuhn, H. W., & Tucker, A. W. (2014). Nonlinear programming. *Traces and emergence of nonlinear programming* (pp. 247–258). Springer.

Laloux, L., Cizeau, P., Potters, M., & Bouchaud, J.-P. (2000). Random matrix theory and financial correlations. *International Journal of Theoretical and Applied Finance*, *3*(03), 391–397.

Ledoit, O., & Wolf, M. (2004). Honey, i shrunk the sample covariance matrix. *The Journal of Portfolio Management*, *30*(4), 110–119.

Lim, K.-P., & Brooks, R. (2011). The evolution of stock market efficiency over time: A survey of the empirical literature. *Journal of Economic Surveys*, *25*(1), 69–108.

Lo, A. W. (2005). Reconciling efficient markets with behavioral finance: The adaptive markets hypothesis. *Journal of investment consulting*, *7*(2), 21–44.

Lubasha, H., Simon, B., Joe, C., Dean, F., Chris, M., Edoardo, P., Anastasios, P., Neil, P., Kedra, N. R., Thomas, S., & Ben, S. (2021). *The $100 trillion machine* (tech. rep.). Boston Consulting Group.

Ma, T., & Jagannathan, R. (2003). Risk reduction in large portfolios: Why imposing the wrong constraints helps. *Capital Markets: Asset Pricing & Valuation*.

MacMahon, M., & Garlaschelli, D. (2013). Community detection for correlation matrices. *arXiv preprint arXiv:1311.1924*.

Man, X., & Chan, E. (2020). The best way to select features? *CoRR*, *abs/2005.12483*. https://arxiv.org/abs/2005.12483

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, *7*(1), 77–91. http://www.jstor.org/stable/2975974

Michaud, R. O. (1989). The markowitz optimization enigma: Is 'optimized'optimal? *Financial analysts journal*, *45*(1), 31–42.

Mike, K., Gerry, H., William, M., Caroline, C., & Ross, T. (2021). *Future of asset management* (tech. rep.). Accenture.

Paranjape-Voditel, P., & Deshpande, U. (2013). A stock market portfolio recommender system based on association rule mining. *Applied Soft Computing*, *13*(2), 1055–1063.

Pearson, K. (1895). Vii. note on regression and inheritance in the case of two parents. *proceedings of the royal society of London*, *58*(347-352), 240–242.

Podgorelec, V., Kokol, P., Stiglic, B., & Rozman, I. (2002). Decision trees: An overview and their use in medicine. *Journal of medical systems*, *26*(5), 445–463.

Potters, M., Bouchaud, J.-P., & Laloux, L. (2005). Financial applications of random matrix theory: Old laces and new pieces. *arXiv preprint physics/0507111*.

Quandl. (2021a). *Sharadar core us fundamentals data*. Retrieved July 28, 2021, from https://www.quandl.com/databases/SF1/data

Quandl. (2021b). *Sharadar equity prices*. Retrieved July 28, 2021, from https://www.quandl.com/databases/SEP/data

Shalev-Shwartz, S., & Zhang, T. (2013). Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, *14*(2).

Stein, C. (2020). Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. *Contribution to the theory of statistics* (pp. 197–206). University of California Press.

Tardivo, G. (2002). Value at risk (var): The new benchmark for managing market risk. *Journal of Financial Management & Analysis*, *15*(1), 16.

Wolff, D., & Echterling, F. (2020). Stock picking with machine learning. *Available at SSRN 3607845*.

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, *5*(2), 241–259.

Zhai, X., Kolesnikov, A., Houlsby, N., & Beyer, L. (2021). Scaling vision transformers. *arXiv preprint arXiv:2106.04560*.

Appendix A

Tables

Name	Description	Duration	Transformation
revenue_qtr	Revenues	Quarterly	Difference
cor_qtr	Cost of Revenue	Quarterly	Difference
sgna_qtr	Selling General and Administrative Expense	Quarterly	Difference
opex_qtr	Operating Expenses	Quarterly	Difference
taxexp_qtr	Income Tax Expense	Quarterly	Difference
consolinc_qtr	Consolidated Income	Quarterly	Difference
netinc_qtr	Net Income	Quarterly	Difference
netinccmn_qtr	Net Income Common Stock	Quarterly	Difference
capex_qtr	Capital Expenditure	Quarterly	Difference
ncff_qtr	Net Cash Flow from Financing	Quarterly	Difference
ncfdebt_qtr	Issuance (Repayment) of Debt Securities	Quarterly	Difference
ncfcommon_qtr	Issuance (Purchase) of Equity Shares	Quarterly	Difference
ncfi_qtr	Net Cash Flow from Investing	Quarterly	Difference
ncfo_qtr	Net Cash Flow from Operations	Quarterly	Difference
ncf_qtr	Net Cash Flow / Change in Cash & Cash Equivalents	Quarterly	Difference
assets_qtr	Total Assets	Quarterly	Difference
cashneq_qtr	Cash and Equivalents	Quarterly	Difference
ppnenet_qtr	Property Plant & Equipment Net	Quarterly	Difference
receivables_qtr	Trade and Non-Trade Receivables	Quarterly	Difference
payables_qtr	Trade and Non-Trade Payables	Quarterly	Difference
liabilities_qtr	Total Liabilities	Quarterly	Difference
equity_qtr	Shareholders Equity	Quarterly	Difference
retearn_qtr	Accumulated Retained Earnings (Deficit)	Quarterly	Difference
accoci_qtr	Accumulated Other Comprehensive Income	Quarterly	Difference
assetsc_qtr	Current Assets	Quarterly	Difference

assetsnc_qtr	Assets Non-Current	Quarterly	Difference
liabilitiesc_qtr	Current Liabilities	Quarterly	Difference
liabilitiesnc_qtr	Liabilities Non-Current	Quarterly	Difference
debt_qtr	Total Debt	Quarterly	Difference
ebt_qtr	Earnings before Tax	Quarterly	Difference
ebit_qtr	Earning Before Interest & Taxes (EBIT)	Quarterly	Difference
ebitda_qtr	Earnings Before Interest Taxes & Depreciation ...	Quarterly	Difference
marketcap_qtr	Market Capitalization	Quarterly	Difference
ev_qtr	Enterprise Value	Quarterly	Difference
invcap_qtr	Invested Capital	Quarterly	Difference
tangibles_qtr	Tangible Asset Value	Quarterly	Difference
fcf_qtr	Free Cash Flow	Quarterly	Difference
gp_qtr	Gross Profit	Quarterly	Difference
opinc_qtr	Operating Income	Quarterly	Difference
workingcapital_qtr	Working Capital	Quarterly	Difference
epsusd_qtr	Earnings per Basic Share (USD)	Quarterly	Percent change
dps_qtr	Dividends per Basic Common Share	Quarterly	Percent change
evebitda_qtr	Enterprise Value over EBITDA	Quarterly	Percent change
evebit_qtr	Enterprise Value over EBIT	Quarterly	Percent change
pe_qtr	Price Earnings (Damodaran Method)	Quarterly	Percent change
pe1_qtr	Price to Earnings Ratio	Quarterly	Percent change
sps_qtr	Sales per Share	Quarterly	Percent change
ps1_qtr	Price to Sales Ratio	Quarterly	Percent change
pb_qtr	Price to Book Value	Quarterly	Percent change
de_qtr	Debt to Equity Ratio	Quarterly	Percent change
currentratio_qtr	Current Ratio	Quarterly	Percent change
price_qtr	Share Price (Adjusted Close)	Quarterly	Percent change
grossmargin_qtr	Gross Margin	Quarterly	Percent change
netmargin_qtr	Profit Margin	Quarterly	Percent change
payoutratio_qtr	Payout Ratio	Quarterly	Percent change
divyield_qtr	Dividend Yield	Quarterly	Percent change
revenue_trailing	Revenues	Yearly	Difference
cor_trailing	Cost of Revenue	Yearly	Difference
sgna_trailing	Selling General and Administrative Expense	Yearly	Difference
opex_trailing	Operating Expenses	Yearly	Difference
intexp_trailing	Interest Expense	Yearly	Difference
taxexp_trailing	Income Tax Expense	Yearly	Difference
consolinc_trailing	Consolidated Income	Yearly	Difference
netinc_trailing	Net Income	Yearly	Difference

capex_trailing	Capital Expenditure	Yearly	Difference
ncfdebt_trailing	Issuance (Repayment) of Debt Securities	Yearly	Difference
ncfcommon_trailing	Issuance (Purchase) of Equity Shares	Yearly	Difference
ncf_trailing	Net Cash Flow / Change in Cash & Cash Equivalents	Yearly	Difference
ebt_trailing	Earnings before Tax	Yearly	Difference
ebit_trailing	Earning Before Interest & Taxes (EBIT)	Yearly	Difference
ebitda_trailing	Earnings Before Interest Taxes & Depreciation ...	Yearly	Difference
equityavg_trailing	Average Equity	Yearly	Difference
assetsavg_trailing	Average Assets	Yearly	Difference
invcapavg_trailing	Invested Capital Average	Yearly	Difference
gp_trailing	Gross Profit	Yearly	Difference
opinc_trailing	Operating Income	Yearly	Difference
epsusd_trailing	Earnings per Basic Share (USD)	Yearly	Percent change
sharefactor_trailing	Share Factor	Yearly	Percent change
evebitda_trailing	Enterprise Value over EBITDA	Yearly	Percent change
evebit_trailing	Enterprise Value over EBIT	Yearly	Percent change
pe_trailing	Price Earnings (Damodaran Method)	Yearly	Percent change
pe1_trailing	Price to Earnings Ratio	Yearly	Percent change
sps_trailing	Sales per Share	Yearly	Percent change
ps1_trailing	Price to Sales Ratio	Yearly	Percent change
pb_trailing	Price to Book Value	Yearly	Percent change
currentratio_trailing	Current Ratio	Yearly	Percent change
price_trailing	Share Price (Adjusted Close)	Yearly	Percent change
roe_trailing	Return on Average Equity	Yearly	Percent change
roic_trailing	Return on Invested Capital	Yearly	Percent change
grossmargin_trailing	Gross Margin	Yearly	Percent change

Table A.2: *Description of the fundamental indicators. We considered quarterly and trailing one-year duration for the earnings report items. The variables in currency unit are made stationary by calculating the percentage change from one period to the next. The others, by taking the first difference. The measures given in a per share basis are multiplied by the number of shares and divided by the enterprise value before applying the difference operation.*

Name	acc	p	r	f1
log_reg_l1	0.506	0.509	0.836	0.632
log_reg_l2	0.502	0.507	0.725	0.596
log_reg_en	0.503	0.507	0.754	0.607
lin_svm	0.504	0.507	0.851	0.636
rbf_svm	0.504	0.507	0.916	0.653
nn	**0.508**	**0.510**	0.854	0.638
dt	*0.499*	*0.504*	0.803	0.620
rf	0.502	0.506	0.817	0.625
gbm	0.503	0.508	0.740	0.602
ens	0.505	0.507	**0.940**	**0.659**
rand	0.499	0.507	*0.508*	*0.507*

Table A.4: *Classification performance measures. The nn achieves the best performance in terms of accuracy and precision. The ensemble model obtains the best performances in terms of recall and f1 score. The baseline rand model performs the worst for all measures considered. This proves empirically that machine learning allows for a better selection of suitable investments than a random selection.*

Name	tot_ret	cagr	win_rate	avg_win	avg_loss	vol	max_dd	var	cvar	sharpe
log_reg_l1_ew	611.98	12.29	60.78	2.14	-2.46	17.14	-54.80	-13.30	-21.92	0.76
log_reg_l1_mv	601.27	12.19	56.52	2.09	-2.22	15.83	-45.18	-11.13	-18.87	0.80
log_reg_l1_shrink	601.97	12.20	57.73	2.09	-2.23	15.67	-45.12	-11.27	-18.79	0.81
log_reg_l1_nco	658.28	12.71	61.51	2.13	-2.45	17.22	-54.44	-13.34	-22.07	0.78
log_reg_l2_ew	576.01	11.95	59.63	2.16	-2.49	20.34	-55.98	-14.36	-25.22	0.66
log_reg_l2_mv	723.84	13.26	57.11	2.13	-2.14	18.21	-45.15	-12.51	-21.03	0.77
log_reg_l2_shrink	721.63	13.24	57.22	2.12	-2.14	18.07	-45.37	-12.38	-20.91	0.78
log_reg_l2_nco	537.50	11.56	59.98	2.15	-2.50	20.42	-56.11	-14.30	-25.33	0.64
log_reg_en_ew	582.22	12.01	59.93	2.16	-2.47	17.27	-54.34	-13.22	-22.02	0.74
log_reg_en_mv	599.69	12.17	55.79	2.14	-2.21	16.02	-45.69	-11.05	-19.06	0.80
log_reg_en_shrink	599.22	12.17	56.03	2.12	-2.21	15.83	-45.72	-10.99	-18.97	0.80
log_reg_en_nco	582.53	12.01	60.66	2.16	-2.48	17.31	-54.49	-13.35	-22.11	0.74
lin_svm_ew	604.16	12.22	59.75	2.13	-2.50	20.29	-56.07	-15.08	-25.18	0.67
lin_svm_mv	754.50	13.51	57.45	2.10	-2.23	17.66	-50.46	-12.40	-21.26	0.80
lin_svm_shrink	759.20	13.54	57.91	2.10	-2.23	17.60	-48.79	-12.24	-21.21	0.81
lin_svm_nco	602.31	12.20	60.32	2.13	-2.49	20.33	-56.11	-15.19	-25.35	0.67
rbf_svm_ew	557.44	11.76	60.78	2.16	-2.50	17.38	-55.22	-13.45	-22.26	0.73
rbf_svm_mv	594.28	12.12	57.13	2.12	-2.27	16.01	-45.25	-11.06	-19.51	0.79
rbf_svm_shrink	587.30	12.06	57.73	2.11	-2.27	15.85	-45.39	-11.06	-19.41	0.80
rbf_svm_nco	582.20	12.01	60.78	2.16	-2.49	17.41	-54.62	-13.34	-22.25	0.74
nn_ew	622.70	12.39	60.38	2.13	-2.48	20.38	-54.80	-15.28	-25.05	0.67
nn_mv	649.11	12.63	58.25	2.08	-2.26	18.75	-49.82	-11.99	-22.26	0.73
nn_shrink	664.49	12.76	58.37	2.08	-2.26	18.62	-49.14	-12.27	-22.10	0.74
nn_nco	566.02	11.85	61.44	2.13	-2.47	20.32	-54.29	-15.05	-25.45	0.65
dt_ew	564.86	11.84	60.54	2.17	-2.49	17.39	-55.22	-13.45	-22.20	0.73
dt_mv	612.88	12.30	57.25	2.15	-2.30	16.15	-45.25	-11.03	-19.40	0.80
dt_shrink	603.24	12.21	58.10	2.13	-2.30	15.97	-45.39	-11.06	-19.31	0.80
dt_nco	590.61	12.09	60.29	2.18	-2.49	17.43	-54.62	-13.77	-22.20	0.74
rf_ew	582.59	12.01	60.44	2.17	-2.52	20.73	-55.24	-14.91	-25.60	0.65
rf_mv	568.64	11.87	56.65	2.14	-2.28	18.05	-45.40	-11.44	-21.77	0.71

rf_shrink	574.55	11.93	56.77	2.14	-2.28	17.90	-45.54	-11.53	-21.67	0.72
rf_nco	640.16	12.55	61.01	2.18	-2.51	20.71	-54.45	-14.75	-25.65	0.67
gbm_ew	580.38	11.99	59.98	2.18	-2.53	20.41	-55.31	-13.96	-25.25	0.66
gbm_mv	611.05	12.28	57.68	2.15	-2.33	18.18	-48.29	-12.40	-21.99	0.73
gbm_shrink	613.52	12.30	58.03	2.13	-2.33	17.90	-48.64	-12.44	-21.72	0.74
gbm_nco	593.08	12.11	59.63	2.19	-2.54	20.42	-55.83	-14.54	-25.30	0.66
ens_ew	545.69	11.64	60.54	2.16	-2.49	17.37	-55.22	-13.45	-22.24	0.72
ens_mv	582.93	12.01	57.00	2.11	-2.27	15.99	-45.25	-10.93	-19.46	0.79
ens_shrink	577.50	11.96	57.98	2.11	-2.27	15.83	-45.39	-10.95	-19.38	0.79
ens_nco	589.44	12.08	61.02	2.16	-2.49	17.42	-54.62	-13.77	-22.24	0.74
rand_ew	627.58	12.43	60.32	2.16	-2.49	19.56	-53.75	-14.43	-24.44	0.70
rand_mv	530.37	11.49	56.88	2.13	-2.31	18.33	-41.75	-12.09	-21.32	0.68
rand_shrink	527.16	11.45	56.77	2.12	-2.31	18.05	-42.39	-12.15	-21.15	0.69
rand_nco	633.19	12.48	60.21	2.16	-2.49	19.56	-53.36	-13.93	-24.35	0.70
bench_ew	597.62	12.15	60.89	2.16	-2.49	19.38	-55.22	-14.29	-24.28	0.69
bench_mv	620.51	12.37	57.68	2.11	-2.27	17.78	-45.25	-11.34	-21.56	0.74
bench_shrink	612.13	12.29	58.60	2.11	-2.27	17.60	-45.39	-11.31	-21.39	0.75
bench_nco	650.81	12.64	61.24	2.16	-2.48	19.41	-54.62	-14.26	-24.27	0.71

Table A.6: *Comparison of the backtest results for all considered selection models and all considered optimization methods against naive random selection and the benchmark index. For each pure performance measure, there are several models that outperform both random and unselected approaches, although this is not the case for all machine learning models. The selection schemes* ***logreg_l1****,* ***logreg_en****,* ***rbf_svm****,* ***dt*** *and* ***ens****. obtain a volatility largely inferior to the others. These are the same models that obtained the best recall scores. The ratio between return and risk is best for the* ***log_reg_l1****,* ***lin_svm*** *and* ***dt*** *when the portfolios are constructed with* ***shrink*** *and* ***mv****. In this respect, all selections perform better than the portfolio composed of randomly selected assets.*

Appendix B

Figures

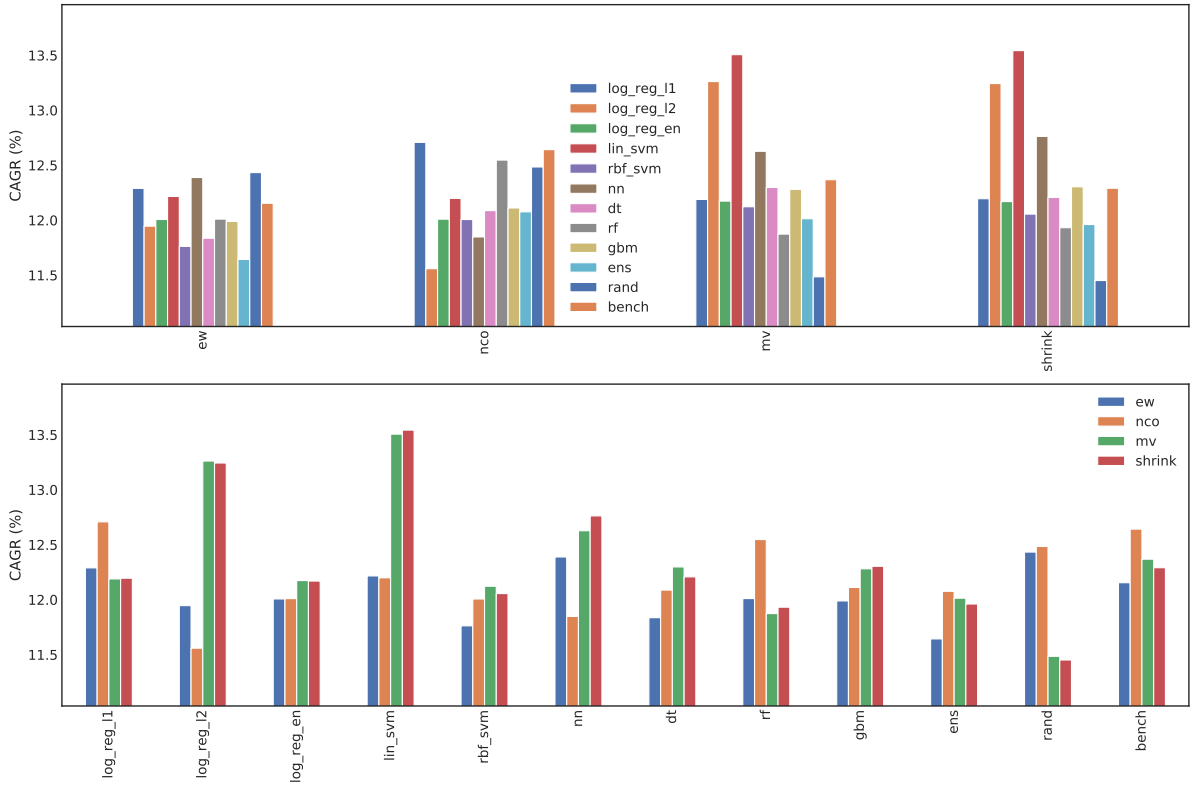

Figure B.1: *Comparison of the cagr. In general, we don't see very big differences in cagr between the proposed methods except for* ***lin_svm*** *and* ***log_reg_l2*** *when they are optimized with* ***mv*** *and* ***shrink****.*

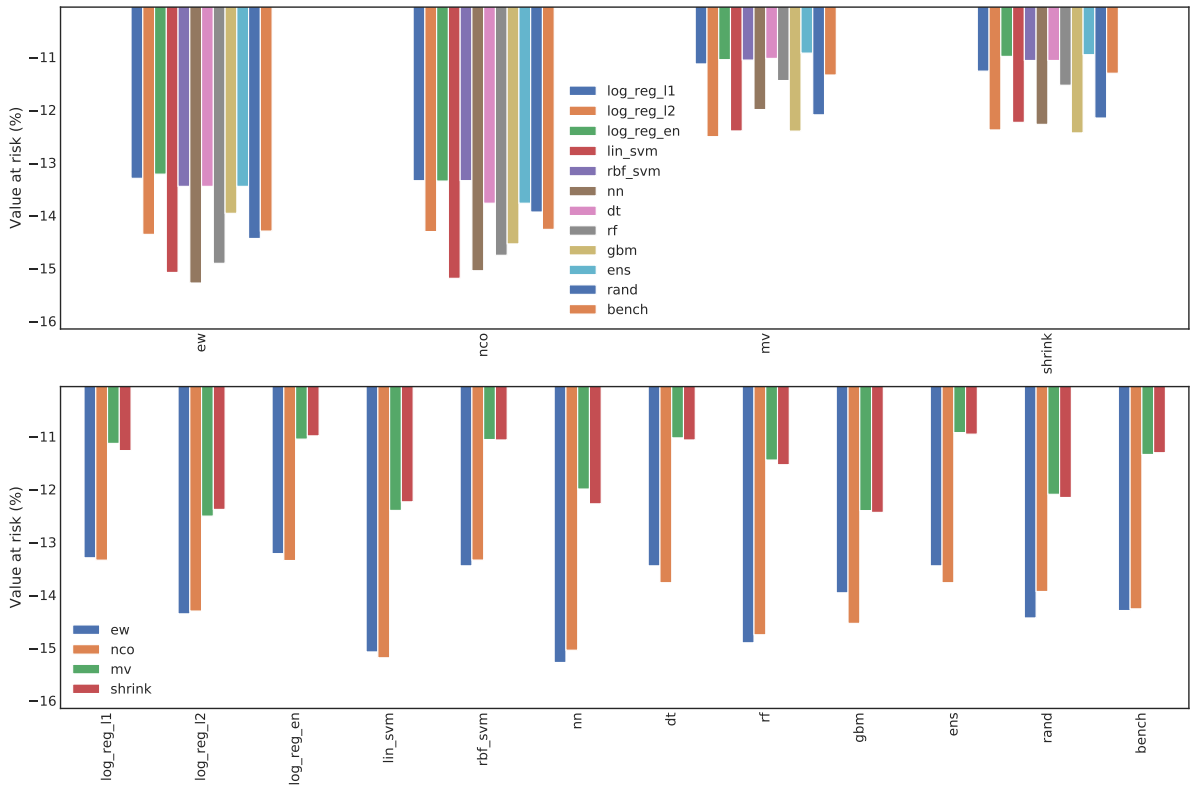

Figure B.2: Comparison of the quarterly value at risk. The value at risk is lowest for the **mv** and **shrink** optimization techniques. The selection schemes **logreg_l1**, **logreg_en**, **rbf_svm**, **dt** and **ens**. obtain a value at risk that is smaller than the other selection methods.

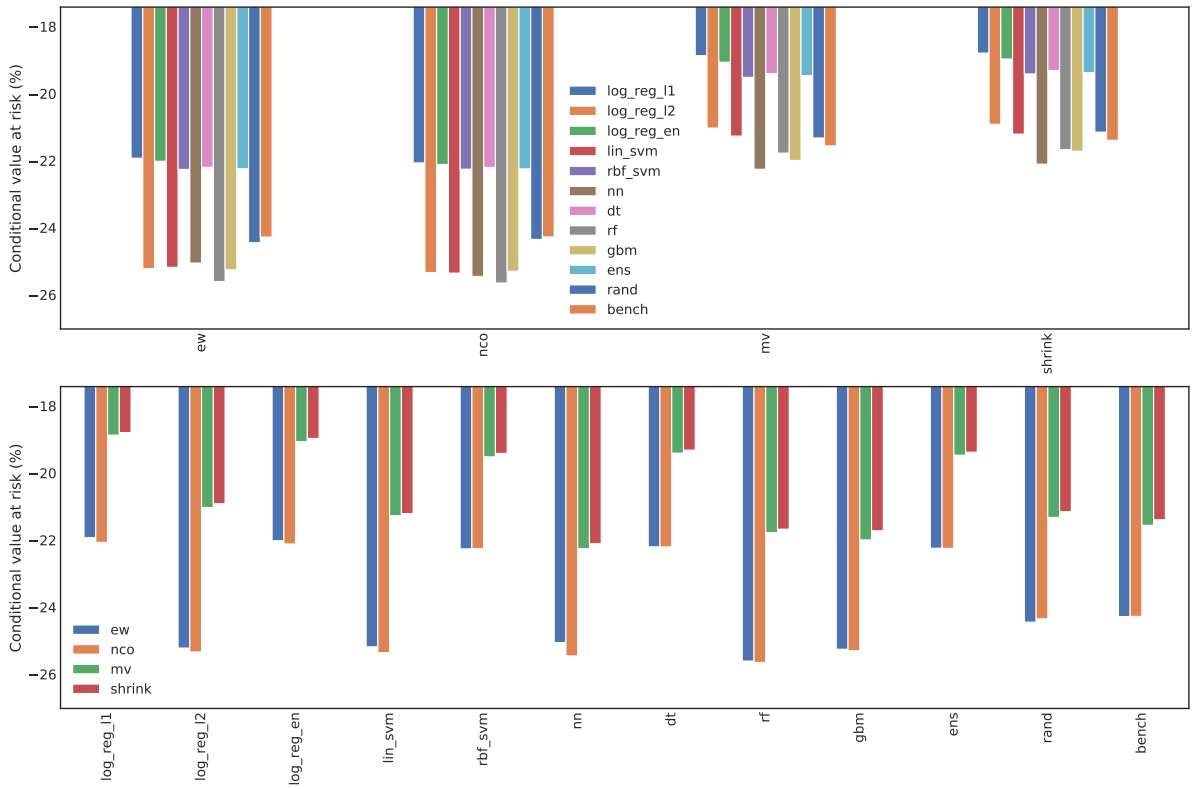

Figure B.3: Comparison of the quarterly conditional value at risk. The conditional value at risk is lowest for the **mv** and **shrink** optimization techniques. The selection schemes **logreg_l1**, **logreg_en**, **rbf_svm**, **dt** and **ens**. obtain a conditional value at risk that is smaller than the other selection methods.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Louvain School of Management
Chaussée de Binche 151, 7000 Mons, Belgique | **www.uclouvain.be/lsm**