Machine Learning in Finance

by

Alexander Windmann

Master's Thesis

submitted to the

Faculty of Mathematics, Computer Science and Natural Sciences
of the RWTH Aachen

August 7, 2020

Supervision by:

Prof. Dr. Stanislaus Maier-Paape
RWTH Aachen

Dr. Mikhail Beketov
Frankfurt School of Finance & Management

Acknowledgement

First of all, my thanks go to my supervisor Professor Maier-Paape, who guided the research and provided helpful advice. With his careful proof-reading, he always pushed me to do my very best, so without him, this thesis would not be the same.
Furthermore, I want to thank Doctor Beketov and Doctor Wittke, who inspired me to write a thesis about machine learning in finance, for putting their trust in me. I also want to thank the many colleagues I had the pleasure to work with at Deloitte.
The same goes for René Brenner and Philipp Kreins from the institute for mathematics at the RWTH, who contributed to this thesis during our weekly meetings.
Of course, I want to thank my parents and my sister, who I can always count on, and my friends. Thanks to my roommates, home office was really enjoyable, as they have always been providing me with a welcome diversion. Finally, I want to thank Laila for joining me on my late-night study sessions and for always supporting me.

Contents

Introduction

In recent decades, technological breakthroughs have occured at an unprecedented pace. Never thought of achievements seem not so distant any more. Already, cars can drive autonomously, at least in a controlled environment. People can interact with their smartphone simply by talking to it via Siri. And computers can detect cancer in CT scans more accurately than any doctor. Many of these spectacular achievements have been realised by machine learning. For ages, humans have dreamt about machines that think and behave like a human being. While there is not a real artificial intelligence yet, giving computers the ability to learn has proven to be successful in many areas where a large amount of data has to be analysed and interpreted. Naturally, the question arises whether it is possible to see similar spectacular results in finance, where vast amounts of data have to be interpreted as fast as possible.

The stock market has seen a tremendous shift towards automated trading during the last decades. The stock exchange floor, where most of the trades were once conducted in person, is now largely a TV set, as trades are usually performed by computers. Quantitative investment funds, in which investment decisions are determined by numerical methods rather than human judgement, became the largest source of institutional trading volume in the American stock market in 2016. In 2019, they accounted for 36% of institutional volume, up from just 18% in 2010, see ["March of the Machines", 2019, p. 19]. Although most of these funds still apply rule-based algorithms, many are pushing automation even further by using machine learning. In a white paper, the quantitative analysts of J. P. Morgan state that "analysts, portfolio managers, traders and CIOs will eventually have to become familiar with Big Data and machine learning approaches to investing", see [Kolanovic, 2017, p. 8].

The goal of this thesis is to give an overview of applications of machine learning in finance. The idea is to cover the whole work process from collecting data to launching a machine learning trading strategy. The thesis is structured as follows: In Chapter 1, there is an introduction to machine learning in general. Then, the process of collecting and structuring data is explained in Chapter 2. In Chapter 3, machine learning techniques are introduced and used in order to create a model that can predict stock price changes. One machine learning technique that can help to detect underlying patterns of financial markets, feature importance, is highlighted in Chapter 4. Then, a trading strategy that applies the machine learning model, a random forest, is introduced in Chapter 5. Moreover, a clustering algorithm is described that allocates the equity to the assets beneficially. Finally, the theory is applied to a real dataset in Chapter 6, where a backtest with post-processing, which uses multiple trend followers, is conducted.

Chapter 1

Types of Machine Learning

Ever since computers have been invented, there was the idea to somehow let machines learn tasks by themselves. In recent years, research in this area has made huge leaps and machine learning is developing rapidly. The use cases are numerous and range from classifying email as spam, recommending items while shopping online to driving an autonomous car. The main idea behind every application is to derive knowledge from data. For a computer program, the process of improving the performance at some task through experience can be defined as 'learning', see [Mitchell, 1997, p. 2].
In general, these algorithms belong to one of the following types of machine learning:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Each of these types will be introduced shortly. For a further discussion, see [Raschka & Mirjalili, 2015, pp. 40-50].

1.1 Supervised learning

In supervised learning, the model is given labelled data in order to find patterns and make predictions about new, unseen data. There are two types of supervised learning: classification and regression.
A classifier has the choice between a few discrete classes, in which similar samples are grouped together. In this context, labelling means specifying the class membership for every sample in the dataset. This way, a program can learn typical features of these classes and use them for class predictions.
Classifying spam emails is a typical use case for supervised learning, see [Hamsapriya et al., 2011, pp. 458-459]. Given a dataset of emails, the program tries to find patterns. For example, there might be certain words that often appear in spam emails. If a new email consists of many of these words, the model predicts that it belongs to the class of spam emails. Theoretically, there could be multiple classes, for example it could be desirable to have a classification in spam, invoices and work related emails.

Other than classifying data with a few, discrete labels, it is also possible to predict a continuous value directly. This supervised learning method is called regression. As for the classifier, the data has to contain information about the response variable that is supposed to be predicted on new data.
In finance, predicting the stock market movement is a popular application area for supervised learning. It is possible to design this as a regression problem, but due to high volatility, predicting an exact price is difficult. Moreover, simply predicting the price ignores its path entirely. A very volatile stock might have a high predicted value, but the risk of suffering a high drawdown is considerable. By classifying assets at a given trading day as good or bad, which will be discussed in Section 2.2, it is possible to bypass these problems. Therefore, the supervised learning models used in this thesis are classifiers.

1.2 Unsupervised learning

While supervised learning depends on labelled data to predict the most likely label or value, unsupervised learning detects patterns without labels.
A popular example is the use of recommender systems. Shopping websites often generate suggestions based on the personal shopping history. This is done by collecting personalized data, like past products, ratings and the age. Then, customers with similar features are clustered by algorithms like the k-means clustering method, see [Kim & Ahn, 2008, p. 1201]. Products that appeal to the peer group are then recommended, as they most likely catch the customer's interest as well.
Unsupervised learning algorithms can be used in finance as well. In the context of asset allocation, clustering algorithms can be used to group similar assets together, which will be discussed in more detail in Subsection 5.3.3. A Principal Component Analysis (PCA) can be used to detect the main features that drive stock price movement, see [López, 2018, p. 118]. Another popular research area is sentiment analysis, where the goal is to extract the sentiment of a piece of text. Used on financial news, it can be used to predict sudden stock price shifts, see [Yadav et al., 2019, p. 313].

1.3 Reinforcement learning

In reinforcement learning, a program has to achieve a goal by interacting with the environment. This interaction can be rewarded or punished, depending on its effect. The program has to learn a way to interact with the environment in such a way that the expected overall reward over time is maximized.
A good example to visualize this concept is an autonomous car. The main goal is to transport the passengers. On the way, the autonomous car has to take a lot of actions, like accelerating, braking or steering. It is hard to prepare the car for every possible scenario by labelling data, as it is done in supervised learning. In a new approach, if the car faces an unknown problem, it has therefore some degree of freedom to act while closely examining the effect of these actions, see [Sallab et al., 2017, p. 71]. This way the car learns from its mistakes and is prepared for a similar situation in the future.

In finance, an application of reinforcement learning are chatbots used on the bank's websites. These can be implemented with reinforcement learning, interacting with the human counterpart to give desired information, see [Papaioannou, 2017, p. 366].

Chapter 2

Data Analysis

The following chapter is dedicated to closely examine the data used for the machine learning models. Financial data has some properties that are unique and have to be accounted for. In a naive approach, one may use standard machine learning methods that seem to produce excellent results, but fail to generalize on new data. The reason mostly lies in the dependency on time and the correlation of multiple different assets.

2.1 Financial Data Structure

A standard assumption for data input in machine learning models is an identically and independent distribution, see [Raschka & Mirjalili, 2015, p. 738]. For example, if the sugar content of grapes is predicted knowing the species, the growing region and the year, grapes with the same characteristics can be expected to have similar results. A hot summer may shift the sugar content up, but that will reflect on all grapes in that region in that year. Therefore, it can be reasonable to assume that the sugar content of grapes has a similar distribution with different parameters. Furthermore, a sample belongs to a specific grape, which is not influenced by the results of other tests, the samples are therefore independent. Such circumstances are good for reliable predictions.
Unfortunately, financial datasets are different. The movement of stock prices is too complex to safely assert an underlying distribution over the entire asset universe. Furthermore, this underlying distribution may change over time, so the data has to be interpreted as a time series. A sample, which in this case is the stock price of an asset on a given day, is therefore highly dependent on prior trading days. Lastly, the price movement is influenced by the price movements of other assets. It is therefore extremely difficult to reliably predict a stock price.
A sample of financial data usually consists of the trading day, a unique ticker symbol to identify an asset and the stock prices for that day. Usually, there is a price for when the stock market opens and closes and for the highest and lowest stock price for the day. At some point in time, there may be a split, meaning that existing shares are divided into smaller shares to boost liquidity. That usually reflects in a sudden price drop of the then smaller share and has to be accounted for. Furthermore, some companies pay dividends to their shareholders. To use the stock price as a performance measure, the raw prices

therefore have to be adjusted. Moreover, often there is an indicator of the daily trading volume for a given stock.
In the following, the dataset will consist of the members of the S&P 500 index from January 1990 to January 2019 and of members of the STOXX Europe 600 index from January 2002 to January 2019, including shares of companies that do not exist anymore due to mergers or bankruptcy. Excluding these is a typical mistake leading to a survivorship bias, where the dataset solely consists of shares of companies that have been successful enough not to go bankrupt. Furthermore, in the dataset used in this thesis, assets are first taken into account when they enter the index. This way, no future knowledge of an asset entering the index is implied.
At the start, a sample consists of a trading day, a ticker symbol and the adjusted close price. To give better predictions, more features have to be added. These features can be either derived from the underlying data, for example the volatility can be estimated given the past adjusted close prices, or from external data like the S&P 500 index and the VIX, which is a volatility index. It is even possible to use non-financial data, like satellite images or social media posts. For an overview of these alternative sources of data see [Kolanovic & Krishnamachari, 2017, pp. 26-50].
Because a feature is used for predictions, it is very important that every feature only contains information that has been available at the time of the given sample. Each of these features is added to the sample vector, resulting in the sample

$$X_i \in \mathbb{R}^{1\times M},\ i \in \{1,\dots,N\},$$

where M is the number of features and N the number of samples. What features to add will be discussed in detail in Chapter 4.
Combining all the samples into a single matrix finally yields the so called feature matrix

$$X \in \mathbb{R}^{N\times M}.$$

2.2 Labelling

As the machine learning models in this thesis use supervised learning, the data have to be labelled. In theory, the model should identify 'good' and 'bad' assets. This classification has to be specified.
There are multiple ways to label financial data. In the following, a couple of labelling methods will be explained and compared. Each of these methods assigns a number, typically -1 or 1, to a single sample. These labels form a vector $y \in \mathbb{Z}^M$, which is added to the feature matrix. The complete dataset can therefore, following the past notations, be expressed as

$$(X, y) \in \mathbb{R}^{N\times(M+1)}.$$

2.2.1 Fixed-time horizon method

The standard labelling method to evaluate the performance of an asset is to look at the future return. Let p_t with $t \in \{1,\dots,T\}$ denote the adjusted close of an asset at the

trading day t. T describes the maximal number of trading days in the dataset. For $t \leq T - h$, the future return with a horizon h or h-day return is defined by

$$r_{t,t+h} = \frac{p_{t+h}}{p_t} - 1.$$

Given a feature matrix X, a sample X_i is assigned a label $y_i \in \{-1, 0, 1\}$ by

$$y_i = \begin{cases} -1 & \text{if } r_{t_i,t_i+h} < -\tau \\ 0 & \text{if } \left|r_{t_i,t_i+h}\right| \leq \tau \\ 1 & \text{if } r_{t_i,t_i+h} > \tau \end{cases}$$

where τ is a pre-defined threshold, t_i is the index of the trading day of the sample X_i and h is the horizon, see [López, 2018, p. 43].

Figure 2.1: The fixed-time horizon labelling method. Depending on the return after 6 months, the label is either 1 (green), 0 (blue) or -1 (red). In the figure, $y_i = 1$.

There are multiple ways to set the threshold. One way is to choose a threshold that leads to classes of roughly the same size, which is a desired property for data for machine learning models. In this case, a threshold of 3% seems to be a good choice for a small horizon of 20 trading days. Apart from the dataset used in this thesis, this threshold seems to apply to other datasets in practice as well, see [Dixon et al., 2016, p. 69]. However, setting a fixed threshold can be problematic, because differences in volatility of assets are not accounted for. Volatile assets are most likely assigned -1 or 1, whereas low-volatility stocks mostly receive 0. The machine learning model will learn this pattern and accurately identify assets with a low volatility, while struggling to identify the correct return sign of volatile assets. A workaround would be to choose a varying threshold, depending on a rolling standard deviation of the past returns.
Suppose there is a sample X_i that describes an asset j at the trading day t_i. For a series of

daily past returns of that asset $(r_{t-1,t})_{t\in\{2,\dots,T\}}$ the standard deviation can be estimated by

$$\sigma_r = \sqrt{\frac{1}{T-2}\sum_{t=2}^{T}\left(r_{t-1,t}-\overline{r}\right)^2},$$

with $\overline{r}$ indicating the arithmetic mean of the return series. However, at the time t_i no future knowledge is allowed to be leaked. Therefore, only the time series for $t \in \{2,\dots,t_i\}$ can be used. Furthermore, techniques like cross-validation, which will be discussed in Subsection 3.1.3, require that there is only a limited window of past observations that is allowed to be used, because a test dataset can predate a training dataset. For the sample X_i, the k-day standard deviation of returns is defined by

$$\sigma_{r,t_i}^{[k]} = \sqrt{\frac{1}{k}\sum_{t=t_i-k}^{t_i}\left(r_{t-1,t}-\overline{r}^{[k]}\right)^2},$$

if $t_i > k$. Of course, the arithmetic mean $\overline{r}^{[k]}$ is calculated with the same rolling window as well. This standard deviation can be set as the varying threshold mentioned earlier. Thus, a sample labelled -1 is considered bad regardless of the asset's specific volatility. Another aspect to consider is that the fixed-time horizon method does not evaluate the path of the stock price. Relying on a single trading day, this method is susceptible to random noise of the price movement signal. Even if, as seen in Figure 2.1, there are big losses in the evaluation period, assets can still be labelled positively.

2.2.2 Triple-barrier method

The triple-barrier method is an advancement of the fixed-time horizon method. As in the latter, there is a time barrier evaluating the return after a certain holding period. If the price path hits this vertical barrier, the sample is labelled 0.
Additionally, there are two horizontal barriers. Firstly, there is a stop-loss limit. When the return crosses a certain negative threshold first, the sample is labelled -1. This threshold can be fixed, for example -3%, or varying and depending on the volatility, as seen in the fixed-time horizon method.
The second barrier is an upper profit-taking barrier. If the price path hits this barrier first, the sample is labelled 1, because the asset performs exceptionally good. Again, this upper barrier can be varying and does not have to be symmetrical to the lower barrier.
Theoretically, any combination of these three barriers is feasible, for example there can be a stop-loss limit and a fixed-time barrier without an upper barrier. It is also possible to omit the neutral class. If the price path touches the vertical barrier, the label then is chosen as -1 or 1 according to the sign of the return at the vertical barrier. For a further discussion, see [López, 2018, pp. 45-47].
The difficulty using this method is to choose good values for the horizontal barriers. In general, classes should be roughly the same size. However, this often leads to an early stop-loss, which is susceptible to sudden random price shifts. Furthermore, setting a long horizon for the fixed-time barrier is computational hard, so labelling takes a lot of time.

Figure 2.2: The triple-barrier labelling method. The label is chosen by the first barrier the price path touches. The upper horizontal barrier (green) indicates 1, the lower horizontal barrier (red) -1 and the vertical barrier (blue) 0. In the figure, $y_i = -1$.

2.2.3 Edge Ratio method

The Edge Ratio method is another advancement ot the fixed-time horizon method. Again, the price return is evaluated after a certain holding period, assigning -1 or 1 depending on the sign of the return. Additionally, the path of the price movement is taken into consideration as well.
The Edge Ratio is designed to evaluate trading strategies, see [Faith, 2007, pp. 65-69]. When the strategy gives signals to buy certain stocks, the Edge Ratio gives information about how profitable this decision would have been. Starting from the time $t_0 \in \{1, \dots, T\}$ the signal has been generated, the maximum adverse excursion (MAE) and the maximum favourable excursion (MFE) are measured for a certain holding period h, provided that $h \leq T - t_0$. Given a series of stock prices relating to a single asset $(p_t)_{t \in \{1,\dots,T-h\}}$, this is done by simply keeping track of the biggest loss and the highest gain for that period:

$$\begin{aligned} \mathrm{MAE}_{t_0}(h) &= p_{t_0} - \min\{p_{t_0}, \dots, p_{t_0+h}\} \geq 0, \\ \mathrm{MFE}_{t_0}(h) &= \max\{p_{t_0}, \dots, p_{t_0+h}\} - p_{t_0} \geq 0. \end{aligned}$$

If the trading strategy only submits one signal to buy a stock at t_0, the Edge Ratio can be calculated by

$$\text{Edge Ratio}(h) = \frac{\mathrm{MFE}_{t_0}(h)}{\mathrm{MAE}_{t_0}(h)}. \tag{2.1}$$

However, if the trading strategy leads to multiple signals, the individual Edge Ratios have to be averaged. To make the MAE and MFE comparable, they are divided by

a volatility measure of the prices. Usually, this volatility measure is the average true range (ATR) of a stock price, see [Giese, 2015, p. 311].
To calculate the ATR, first the true high and true low of a stock price have to be assessed. As mentioned earlier, a dataset often contains the highest and lowest value of the stock price at a given trading day t. The true high is the maximum of the highest price today and the close price of the previous trading day:

$$p_t^{\text{true high}} = \max\left\{p_t^{\text{high}}, p_{t-1}^{\text{close}}\right\}.$$

Similarly, the true low can be calculated:

$$p_t^{\text{true low}} = \min\left\{p_t^{\text{low}}, p_{t-1}^{\text{close}}\right\}.$$

Finally, the ATR for k days at the trading day t_0 can be calculated by averaging the differences of the true highs and lows:

$$\text{ATR}_{t_0}^{[k]} = \frac{1}{k}\sum_{t=t_0-k+1}^{t_0}\left(p_t^{\text{true high}} - p_t^{\text{true low}}\right).$$

Suppose the trading strategy gives multiple signals $j \in \{1,\dots,n\}$ to buy stocks. To derive the Edge Ratio for the signal j at the trading day t_j, the MAE and the MFE are scaled by

$$\text{MAE}_{t_j}^*(h,k) = \frac{\text{MAE}_{t_j}(h)}{\text{ATR}_{t_j}^{[k]}} \quad \text{and} \quad \text{MFE}_{t_j}^*(h,k) = \frac{\text{MFE}_{t_j}(h)}{\text{ATR}_{t_j}^{[k]}}.$$

Finally, the Edge Ratio can be derived by

$$\text{Edge Ratio}(h,k) = \frac{\sum_{j=1}^{n}\text{MFE}_{t_j}^*(h,k)}{\sum_{j=1}^{n}\text{MAE}_{t_j}^*(h,k)}.$$

In the context of labelling, calculating the Edge Ratio can be simplified. As the label is specific for every sample, only one asset has to be examined at a time. Therefore, there is no need to scale the MAE and the MFE. Hence, in the following, the Edge Ratio will refer to Equation (2.1) that describes the ratio of the unscaled MFE and MAE for one asset.
The main idea of the Edge Ratio labelling method is to update the fixed-time horizon method with the Edge Ratio as a performance measure for the assets' price path. However, in contrast to the previous methods, there is no 'neutral' 0 class of samples with ambiguous price development. This is done intentionally, because such a class is seldom necessary. In Chapter 6, supervised learning classifiers are used to identify the assets that are most likely to have a good future price development. This information is not coded in the label alone. With every prediction, the models also specify the probability that the sample belongs to the chosen class, more on that later in the Sections 3.3 and 3.4. A prediction of a positive label with low confidence therefore serves the same purpose as a neutral label: it indicates an ambiguous price development. Some of the samples

with an ambiguous price movement are labelled -1, while others are labelled 1. This is apparent when samples of a single asset are isolated. At some point in time, the samples switch their label, while their features have only changed gradually. Therefore, it is hard to differentiate between those samples. When the classifier has to predict the class of a similar sample, it can do so with a low confidence only. In fact, even the author of the triple-barrier labelling method advises on omitting the neutral class, see [López, 2018, p. 69]. Hence, the Edge Ratio labelling method introduced in this thesis relies solely on a positive and a negative label. Of course, the definition can be easily adjusted by adding a neutral class for average returns or for Edge Ratios close to 1.
To get a positive label, a sample should have both a good future price path and a positive future return. An Edge Ratio of 1 indicates a neutral price path. Samples that possess a lower Edge Ratio should be labelled negatively, because at some point the loss has outweighed the profit. Intuitively, it might be reasonable to set a high Edge Ratio threshold to specifically target high performing assets. However, this comes at a cost. To see this, a classifier is given the task to predict the Edge Ratio label with different Edge Ratio thresholds. It can be shown that a high Edge Ratio threshold lowers the predicted probabilities of positive labels, which are very important in a financial context. More importantly, the precision of the predictions, which is defined in Subsection 3.1.4, decreases. This can be seen in Figure 2.3, where every sample of the test set is categorized depending on its predicted and its true label in so called confusion matrices.

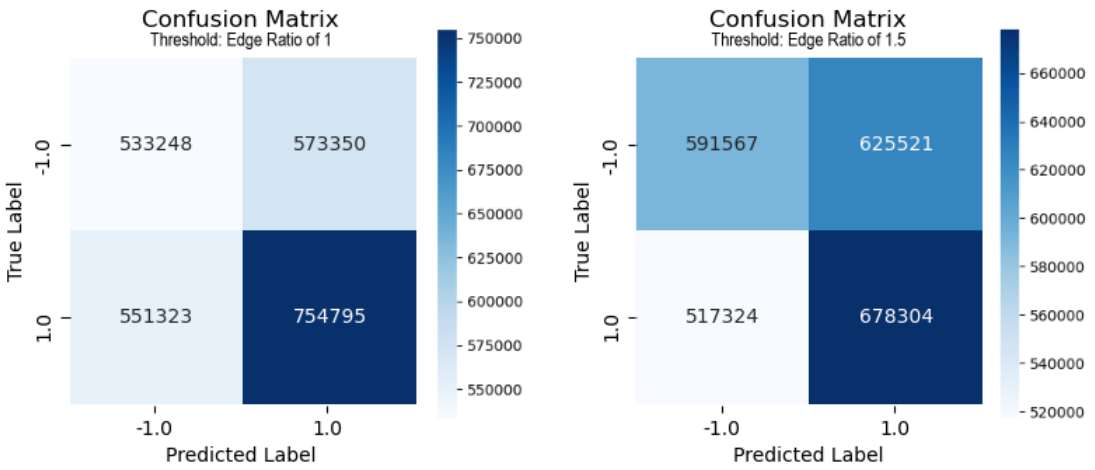

Figure 2.3: Two confusion matrices categorizing samples depending on their 120-day Edge Ratio labels. Elevating the threshold decreases the precision while improving the prediction of negatively labelled samples.

In short, the smaller the positive class is, the harder it is to predict positively labelled samples, as the positively labelled training dataset shrinks. However, increasing the size of the positive class comes at the cost of less desirable predictions, because more samples with a poor future performance are labelled positively as well. This should be kept

in mind while also trying to create roughly even sized classes overall, although small unbalances can be fixed easily by giving underrepresented samples more weight, see Section 3.2.
As the algorithm used in Chapter 6 only invests in assets if the associated samples are labelled positively with a high predicted probability, a high precision is essential. The prediction of negatively labelled samples is of lower importance. Therefore, the Edge Ratio threshold for the Edge Ratio labelling method in this thesis is set to the neutral minimum of 1. If the prediction of bad assets is important, for example if an investor tries to bet on falling prices, decreasing the size of the positive class becomes useful, as predictions of negatively labelled samples become better.
Additionally to the Edge Ratio as a performance measure for the path, the return at a fixed-time horizon should be checked, because an asset can have a good Edge Ratio and still perform poorly. It is possible for the price to rise sharply, only to steadily fall until the return on the fixed-time horizon is negative. In this case, the Edge Ratio is bigger than 1 while a loss is realized.
Setting the return threshold for positively labelled samples follows the same logic as the Edge Ratio threshold. The higher the threshold is set, the smaller the positive class gets and the less reliable predictions of positive labels become. Hence, a minimal reasonable return threshold of 0 is set. This way, an investment based on positive labels is predicted to at least not make a loss. In Chapter 6, this strategy proves to be successful.
The final labelling method for the label y_i of a sample X_i, which is depicted in Figure 2.4, consists of the following steps:

1. Calculate the MAE and the MFE of X_i at t_i over a period specified by the labelling horizon h, with t_i indicating the index of the trading day of X_i.

2. Calculate the Edge Ratio by dividing the MFE by the MAE.

3. Calculate the return at the fixed-time horizon r_{t_i,t_i+h}.

4. Assign $y_i = 1$, if the Edge Ratio is bigger than 1 and $r_{t,t+h}$ is bigger than 0.
   Else, assign $y_i = -1$.

As with the other labelling methods, a labelling horizon has to be chosen. Again, this parameter influences the class distribution. When using a small horizon h, the price movement is prone to random shifts. The h-day return is often negative, therefore -1 is assigned most often, see Table 2.1. The higher the horizon is chosen, the bigger the positive class becomes, as the long-term market movement is mostly upward. Fortunately, this small class unbalance in favour of the positive class is desirable, as described earlier.

Label	Labelling Horizon (in Days)					
	10	20	40	60	90	120
-1	55.41%	52.79%	50.26%	48.63%	46.84%	45.55%
1	44.59%	47.21%	49.74%	51.37%	53.16%	54.45%

Table 2.1: Label distribution of the Edge Ratio labelling method on the S&P 500 dataset.

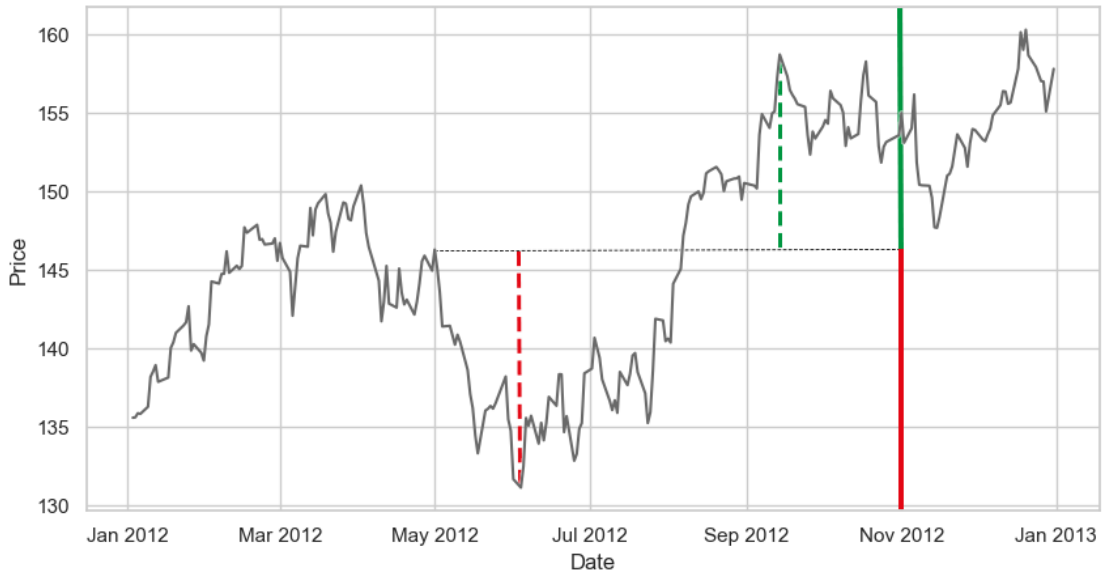

Figure 2.4: The Edge Ratio labellig method. The label is 1, if the MAE (red, dashed) is smaller than the MFE (green, dashed) and the return after 6 months is positive (green, solid). Otherwise, it is -1. In the figure, $y_i = -1$.

The Edge Ratio labelling method is especially useful when using a big labelling horizon, because it is computational fast and random price shifts at the fixed-time horizon are evened out by taking the price path into account. In the case study in Chapter 6, the reallocation of the portfolio is restricted to four times a year to limit transaction costs. Therefore, long labelling horizons are necessary to give a good prediction about long-term price development. Hence, the Edge Ratio labelling method is used in the rest of this thesis. The optimal labelling horizon will be analysed in Subsection 6.1.2.

Chapter 3

Modelling

After the dataset has been set up and labelled, the next step is to build a model that can predict these labels. In order to do that, evaluation methods to compare different models have to be defined. Then, an actual model is introduced and improved through machine learning techniques.

3.1 Model Evaluation

The problem of supervised learning is to measure the performance of a model. In practice, what matters is the prediction accuracy on new data. However, labelling these new data is often not possible, so there is no possibility to check whether the prediction is correct directly. This is especially true for financial data, as the label depends on the price movement in the future and may not be available for a long time.

3.1.1 Bias-Variance Trade-off

An important issue to keep in mind when working with machine learning models is the bias-variance trade-off. In general, there are three errors to consider, see [López, 2018, pp. 93-94]:

1. **Bias:** If a model is not complex enough, it fails to detect the relationship between certain features and the label. The model is 'underfit', as it cannot recognize important patterns in the dataset.

2. **Variance:** If a model is too complex, it suffers from high variance. The model is very sensitive to small changes in the training data and predictions vary a lot depending on the samples used for training. Consequently, the model generalizes bad on new, unseen data. The model is 'overfit' and interprets a lot of random noise as a pattern.

3. **Irreducible error:** When observing values, there is always a variance due to randomness or measurement errors. This error cannot be reduced, as real world values seldom follow an underlying function in a perfect way.

To make this concept clear, it is helpful to closely examine a common area of machine learning: regression.
Suppose there are $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$-valued random variables $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{\mathcal{E}}$ on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. $\mathcal{B}(\mathbb{R})$ denotes the Borel algebra on $\mathbb{R}$. Moreover, suppose there is an underlying relationship between those random variables, which can be described with a measurable function $f : (\mathbb{R}, \mathcal{B}(\mathbb{R})) \to (\mathbb{R}, \mathcal{B}(\mathbb{R}))$ as

$$\boldsymbol{Y} = f(\boldsymbol{X}) + \boldsymbol{\mathcal{E}}, \tag{3.1}$$

with $\mathrm{E}\left[\boldsymbol{\mathcal{E}}\right] = 0$ and $\mathrm{E}\left[\boldsymbol{\mathcal{E}}^2\right] < \infty$. The random variable $\boldsymbol{\mathcal{E}}$ describes random noise. It is stochastically independent of $\boldsymbol{X}$ and describes an error induced by measurement errors. Furthermore, suppose that the second moment of $f(\boldsymbol{X})$ is finite, so

$$\mathrm{E}\left[(f(\boldsymbol{X}))^2\right] < \infty.$$

In essence, this implies that the expected value and the variance of $\boldsymbol{Y}$ are finite.
The goal of regression is to find a predictor for f that minimizes the expected loss. In order to accurately describe this expected loss, first the notation has to be clarified.

Notation

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. For a random vector $\boldsymbol{Z} : (\Omega, \mathcal{F}) \to (\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$, the probability distribution $P_{\boldsymbol{Z}}$, which is a probability measure by itself, can be defined by

$$P_{\boldsymbol{Z}} : \mathcal{B}(\mathbb{R}^n) \to [0,1], \quad B \mapsto P_{\boldsymbol{Z}}(B) = \mathbb{P}\big(\{\omega \in \Omega : \boldsymbol{Z}(\omega) \in B\}\big) = \mathbb{P}\left(\boldsymbol{Z}^{-1}(B)\right).$$

The probability distribution can be used to calculate the expected value of $\boldsymbol{Z}$. The general transport formula states that, given a random vector $\boldsymbol{Z} : (\Omega, \mathcal{F}) \to (\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$, a measurable function $f : (\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n)) \to (\mathbb{R}, \mathcal{B}(\mathbb{R}))$ is $P_{\boldsymbol{Z}}$-integrable, iff $f \circ \boldsymbol{Z}$ is $\mathbb{P}$-integrable. In this case,

$$\mathrm{E}\left[f(\boldsymbol{Z})\right] = \int_{\Omega} f(\boldsymbol{Z}(\omega)) d\mathbb{P}(\omega) = \int_{\mathbb{R}} f(z) dP_{\boldsymbol{Z}}(z)$$

holds, see [Oloff, 2017, pp. 122-124]. In the regression scenario explained, this formula is valid, because f is Borel measurable by design and

$$\mathrm{E}\left[|f(\boldsymbol{X})|\right] \leq \sqrt{\mathrm{E}\left[(f(\boldsymbol{X}))^2\right]} < \infty.$$

In the last step, Jensen's inequality was used.
Since $\boldsymbol{X}$ and $\boldsymbol{\mathcal{E}}$ are defined on the same probability space, the distribution for the random vector $(\boldsymbol{X}, \boldsymbol{\mathcal{E}})$ can be defined as well, see [Oloff, 2017, p. 160]:

$$P_{(\boldsymbol{X},\boldsymbol{\mathcal{E}})} : \mathcal{B}(\mathbb{R}^2) \to [0,1], \quad B_1 \times B_2 \mapsto P_{(\boldsymbol{X},\boldsymbol{\mathcal{E}})}\left(B_1 \times B_2\right) = \mathbb{P}\left((\boldsymbol{X}, \boldsymbol{\mathcal{E}})^{-1}(B_1 \times B_2)\right).$$

By construction, the random variables $\boldsymbol{X}$ and $\boldsymbol{\mathcal{E}}$ are stochastically independent, therefore

$$P_{(\boldsymbol{X},\boldsymbol{\mathcal{E}})}\left(B_1 \times B_2\right) = P_{\boldsymbol{X}}\left(B_1\right) \cdot P_{\boldsymbol{\mathcal{E}}}\left(B_2\right)$$

for all $B_1 \times B_2 \in \mathcal{B}(\mathbb{R}^2)$, see [Oloff, 2017, p. 161].
In order to examine the underlying relationship f, realizations of $\boldsymbol{X}$ and $\boldsymbol{Y}$ are collected in the dataset $d_{all} = \{(x_1, y_1, \ldots, (x_N, y_N)\}$. A single sample can be interpreted as the realization of the random vector

$$\boldsymbol{D} : (\Omega, \mathcal{F}) \to \left(\mathbb{R}^2, \mathcal{B}\left(\mathbb{R}^2\right)\right), \quad \omega \mapsto \boldsymbol{D}(\omega) = (\boldsymbol{X}(\omega), \boldsymbol{Y}(\omega)).$$

As it has been shown before for $(\boldsymbol{X}, \boldsymbol{\mathcal{E}})$, the distribution of $\boldsymbol{D}$ is $P_{\boldsymbol{D}} = P_{(\boldsymbol{X},\boldsymbol{Y})}$. Now suppose that multiple samples that are independent and identically distributed are collected in a set. In the following, this dataset will be modelled as a stochastic process. In order to do that, first a probability measure is defined. For every permutation $\pi : \{1, \ldots, N\} \to \{1, \ldots, N\}$ and for all Borel sets $B_{i,1} \times B_{i,2} \in \mathcal{B}(\mathbb{R}^2)$ with $i \in \{1, \ldots, N\}$ define

$$P_{(\pi(1),\ldots,\pi(N))} : \bigotimes_{i=1}^{N} \left(\mathcal{B}\left(\mathbb{R}^2\right)\right) \to [0,1], \quad \bigtimes_{i=1}^{N} \left(B_{\pi(i),1} \times B_{\pi(i),2}\right) \mapsto \prod_{i=1}^{N} P_{\boldsymbol{D}}\left(B_{\pi(i),1} \times B_{\pi(i),2}\right),$$

where $\bigotimes_{i=1}^{N} \left(\mathcal{B}\left(\mathbb{R}^2\right)\right) = \mathcal{B}\left(\mathbb{R}^{2N}\right)$ denotes the tensor-product σ-algebra. This probability measure fulfils two properties. First, because the product is commutative, for every permutation $\pi : \{1, \ldots, N\} \to \{1, \ldots, N\}$ and for all Borel sets $B_{i,1} \times B_{i,2} \in \mathcal{B}(\mathbb{R}^2)$ with $i \in \{1, \ldots, N\}$ it follows that

$$\begin{aligned}
&P_{(\pi(1),\ldots,\pi(N))}\left(B_{\pi(1),1} \times B_{\pi(1),2} \times \ldots \times B_{\pi(N),1} \times B_{\pi(N),2}\right) \\
&= \prod_{i=1}^{N} P_{\boldsymbol{D}}(B_{\pi(i),1} \times B_{\pi(i),2}) \\
&= \prod_{i=1}^{N} P_{\boldsymbol{D}}(B_{i,1} \times B_{i,2}) \\
&= P_{(1,\ldots,N)}\left(B_{1,1} \times B_{1,2} \times \ldots \times B_{N,1} \times B_{N,2}\right).
\end{aligned}$$

Second, for $\bigtimes_{i=1}^{N-1} (B_{i,1} \times B_{i,2}) \in \bigotimes_{i=1}^{N-1} \left(\mathcal{B}\left(\mathbb{R}^2\right)\right)$, it can be shown that

$$\begin{aligned}
P_{(1,\ldots,N-1)}\left(\bigtimes_{i=1}^{N-1} (B_{i,1} \times B_{i,2})\right) &= \prod_{i=1}^{N-1} P_{\boldsymbol{D}}\left(B_{i,1} \times B_{i,2}\right) \\
&= \prod_{i=1}^{N-1} P_{\boldsymbol{D}}\left(B_{i,1} \times B_{i,2}\right) \cdot P_{\boldsymbol{D}}\left(\mathbb{R}^2\right) \\
&= P_{(1,\ldots,N)}\left(\bigtimes_{i=1}^{N-1} (B_{i,1} \times B_{i,2}) \times \mathbb{R}^2\right).
\end{aligned}$$

Therefore, using the Kolmogorov extension theorem, see [Oloff, 2017, p. 213], it follows that there exists a stochastic process

$$\boldsymbol{\mathcal{D}}_N = \{\boldsymbol{D}_i\}_{i \in \{1,\ldots,N\}}$$

for independent and identically distributed random vectors $\boldsymbol{D}_i = (\boldsymbol{X}_i, \boldsymbol{Y}_i) \sim P_{\boldsymbol{D}}$ with the probability distribution

$$P_{\boldsymbol{\mathcal{D}}_N} = P_{(1,\dots,N)}.$$

This stochastic process models the whole dataset that is used in this regression scenario. That means that d_{all} can be interpreted as a realization of $\boldsymbol{\mathcal{D}}_N$.

Modelling a predictor

The goal of regression is to use a training dataset $d \subset d_{all}$ of size $n < N$ to estimate f with a measurable function

$$\hat{f}_d : (\mathbb{R}, \mathcal{B}(\mathbb{R})) \to (\mathbb{R}, \mathcal{B}(\mathbb{R})),$$

which is called a predictor of f. The dataset d_{all} is split into a training subset d and a test subset d_{test}, which will be discussed in detail in Subsection 3.1.2. The samples in the dataset are independent and identically distributed. As it has been shown for $\boldsymbol{\mathcal{D}}_N$, the distribution of N samples is simply the product of the distributions of the individual samples. Therefore, the distribution of the training dataset can be described by the stochastic process $\boldsymbol{\mathcal{D}}_N$ that is stopped after n samples have been drawn, which is denoted by $\boldsymbol{\mathcal{D}}_n$. Hence, d can be interpreted as a realization of $\boldsymbol{\mathcal{D}}_n$.
The exact definition of $\hat{f}_d$ depends on the actual model that has been chosen. For example, $\hat{f}_d$ can be modelled as an affine function, which is the scenario described in linear regression. In this case, the training dataset is used to derive the optimal slope and intercept of $\hat{f}_d$. Essentially, more complex models like neural networks work similarly, although there are potentially millions of parameters that have to be set. The model specifies how to use the samples in the training dataset to build the predictor $\hat{f}_d$. In order to derive a universal decomposition for a measurable predictor $\hat{f}_d$, the specific definition of $\hat{f}_d$ is kept vague intentionally.
To evaluate a possible predictor $\hat{f}_d$, a loss function is applied on the samples of the test set. The predictor $\hat{f}_d$ estimates f very well if the loss is small, thus it is modelled in a way that the loss function is minimized. In regression, a popular loss function is the mean squared error

$$\frac{1}{m} \sum_{i=1}^{m} \left(y_i - \hat{f}_d(x_i) \right)^2,$$

for $(x_i, y_i) \in d_{\text{test}}$ with $|d_{\text{test}}| = m = N - n$. Suppose for a moment that the test dataset were infinite in size. In that case, the strong law of large numbers states that this average converges almost surely to the expected loss, which is defined as the limit

$$\frac{1}{m} \sum_{i=1}^{m} \left(y_i - \hat{f}_d(x_i) \right)^2 \overset{a.s.}{\to} \mathrm{E}\left[\left(\boldsymbol{Y} - \hat{f}_d(\boldsymbol{X}) \right)^2 \right],$$

for $m \to \infty$. Unfortunately, the test dataset is finite, so empirically, the expected loss can only be estimated with the mean squared error. Under the assumption that this expected loss is finite, which is a reasonable assumption to make for the predictor $\hat{f}_d$,

the expected loss can be decomposed:

$$\begin{aligned}\mathrm{E}\left[\left(\boldsymbol{Y}-\hat{f}_d(\boldsymbol{X})\right)^2\right] &\overset{(3.1)}{=} \mathrm{E}\left[\left(f(\boldsymbol{X})+\boldsymbol{\mathcal{E}}-\hat{f}_d(\boldsymbol{X})\right)^2\right] \\ &= \int_{\mathbb{R}^2}\left(f(x)+\varepsilon-\hat{f}_d(x)\right)^2 dP_{(\boldsymbol{X},\boldsymbol{\mathcal{E}})}(x,\varepsilon) \\ &= \int_{\mathbb{R}}\int_{\mathbb{R}}\left(f(x)+\varepsilon-\hat{f}_d(x)\right)^2 dP_{\boldsymbol{\mathcal{E}}}(\varepsilon)dP_{\boldsymbol{X}}(x).\end{aligned}$$

Again, the transport formula can be applied because the integrand is measurable as a composition of measurable functions and the first integral exists. Furthermore, because the integrand is non-negative and $\boldsymbol{X}$ and $\boldsymbol{\mathcal{E}}$ are stochastic independent, Tonelli's theorem can be applied in the last step, see [Henze, 2019, p. 351].
For different training datasets, the predictor and therefore the expected error varies, because d is itself a realization of the stochastic process $\boldsymbol{\mathcal{D}}_n$. If a training set d is chosen, it is impossible to be certain that this realization of $\boldsymbol{\mathcal{D}}_n$ yields the best predictor $\hat{f}_d$. After all, the training set can be updated with new data, therefore improving or impairing the predictor.
To make the dependency on $\boldsymbol{\mathcal{D}}_n$ clear, the predictor can be generalized. Fix a realization x of $\boldsymbol{X}$. Then the predicted value for x in dependence of the training set d can be defined as the measurable function

$$\hat{f}_\cdot(x) : \left(\bigtimes_{i=1}^{n}\mathbb{R}^2,\ \bigotimes_{i=1}^{n}\left(\mathcal{B}\left(\mathbb{R}^2\right)\right)\right) \to \left(\mathbb{R},\mathcal{B}(\mathbb{R})\right), \quad d \mapsto \hat{f}_d(x).$$

The combined predictor

$$\hat{f} : \left(\left(\bigtimes_{i=1}^{n}\mathbb{R}^2\right)\times\mathbb{R},\ \left(\bigotimes_{i=1}^{n}\left(\mathcal{B}\left(\mathbb{R}^2\right)\right)\right)\otimes\mathcal{B}\left(\mathbb{R}\right)\right) \to \left(\mathbb{R},\mathcal{B}(\mathbb{R})\right) \quad (d,x)\mapsto \hat{f}_d(x)$$

has to be measurable as well. Moreover, as a last requirement, the second moment of the predicted value has to be finite:

$$\mathrm{E}\left[\left(\hat{f}_{\boldsymbol{\mathcal{D}}_n}(\boldsymbol{X})\right)^2\right] = \int_{\Omega}\left(\hat{f}_{\boldsymbol{\mathcal{D}}_n(\omega)}(\boldsymbol{X}(\omega))\right)^2 d\mathbb{P}(\omega) < \infty,$$

whereby the expected value is derived in regards to both the training dataset ($\boldsymbol{\mathcal{D}}_n$) and the test dataset ($\boldsymbol{X}$). This ensures that both the expected prediction and the variance of the predictor exist.

Decomposition

Now that the predictor has been modelled, the true expected loss can be defined and decomposed. By construction, the loss function

$$\begin{aligned}l : &\left(\left(\bigtimes_{i=1}^{n}\mathbb{R}^2\right)\times\mathbb{R}\times\mathbb{R},\ \left(\bigotimes_{i=1}^{n}\left(\mathcal{B}\left(\mathbb{R}^2\right)\right)\right)\otimes\mathcal{B}\left(\mathbb{R}\right)\otimes\mathcal{B}\left(\mathbb{R}\right)\right) \to \left(\mathbb{R},\mathcal{B}(\mathbb{R})\right) \\ &(d,x,y)\mapsto l(d,x,y) = \left(y-\hat{f}_d(x)\right)^2\end{aligned}$$

is integrable and measurable as a composition of measurable functions, therefore the general transport formula can be applied to the expected loss:

$$\begin{aligned}
\mathrm{E}\left[\left(\boldsymbol{Y}-\hat{f}_{\boldsymbol{\mathcal{D}}_n}(\boldsymbol{X})\right)^2\right] &\overset{(3.1)}{=} \mathrm{E}\left[\left(f(\boldsymbol{X})+\boldsymbol{\mathcal{E}}-\hat{f}_{\boldsymbol{\mathcal{D}}_n}(\boldsymbol{X})\right)^2\right] \\
&= \int_{\Omega}\left(f(\boldsymbol{X}(\omega))+\boldsymbol{\mathcal{E}}(\omega)-\hat{f}_{\boldsymbol{\mathcal{D}}_n(\omega)}(\boldsymbol{X}(\omega))\right)^2 d\mathbb{P}(\omega) \\
&= \int_{\mathbb{R}^{2+2n}}\left(f(x)+\varepsilon-\hat{f}_d(x)\right)^2 dP_{(\boldsymbol{X},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{D}}_n)}(x,\varepsilon,d).
\end{aligned}$$

By construction, all the samples in the dataset are independent and identically distributed. In particular, the training set (modelled by $\boldsymbol{\mathcal{D}}_n$) is independent of the test set (modelled by $(\boldsymbol{X},\boldsymbol{\mathcal{E}})$). Therefore, the joint probability distribution can be split:

$$\begin{aligned}
P_{(\boldsymbol{X},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{D}}_n)}\left(B_1\times B_2\times\bigtimes_{i=1}^{n}(B_{i,1}\times B_{i,2})\right) &= P_{(\boldsymbol{X},\boldsymbol{\mathcal{E}})}(B_1\times B_2)\cdot P_{\boldsymbol{\mathcal{D}}_n}\left(\bigtimes_{i=1}^{n}(B_{i,1}\times B_{i,2})\right) \\
&= P_{\boldsymbol{X}}(B_1)\cdot P_{\boldsymbol{\mathcal{E}}}(B_2)\cdot P_{\boldsymbol{\mathcal{D}}_n}\left(\bigtimes_{i=1}^{n}(B_{i,1}\times B_{i,2})\right),
\end{aligned}$$

for all $B_1\times B_2\times\bigtimes_{i=1}^{n}(B_{i,1}\times B_{i,2})\in\mathcal{B}\left(\mathbb{R}^{2+2n}\right)$.
Hence, using the fact that the loss function l is non-negative, Tonelli's theorem can be applied again:

$$\begin{aligned}
\mathrm{E}\left[\left(\boldsymbol{Y}-\hat{f}_{\boldsymbol{\mathcal{D}}_n}(\boldsymbol{X})\right)^2\right] &\overset{(3.1)}{=} \int_{\mathbb{R}^{2+2n}}\left(f(x)+\varepsilon-\hat{f}_d(x)\right)^2 dP_{(\boldsymbol{X},\boldsymbol{\mathcal{E}},\boldsymbol{\mathcal{D}}_n)}(x,\varepsilon,d) \\
&= \int_{\mathbb{R}}\int_{\mathbb{R}}\int_{\mathbb{R}^{2n}}\left(f(x)+\varepsilon-\hat{f}_d(x)\right)^2 dP_{\boldsymbol{\mathcal{D}}_n}(d)dP_{\boldsymbol{\mathcal{E}}}(\varepsilon)dP_{\boldsymbol{X}}(x).
\end{aligned}$$

First, the innermost integral is decomposed. For this purpose, fix realizations x and ε of $\boldsymbol{X}$ and $\boldsymbol{\mathcal{E}}$, which is equivalent to selecting a single sample (x,y) of the test set, because $\varepsilon=y-f(x)$. Both the expected loss and the second moment of the predicted value are finite by construction, thus without loss of generality let x and ε be such that

$$\int_{\mathbb{R}^{2n}}\left(f(x)+\varepsilon-\hat{f}_d(x)\right)^2 dP_{\boldsymbol{\mathcal{D}}_n}(d)=\mathrm{E}\left[\left(f(x)+\varepsilon-\hat{f}_{\boldsymbol{\mathcal{D}}_n}(x)\right)^2\right]<\infty$$

and $\mathrm{E}\left[\left(\hat{f}_{\mathcal{D}_n}(x)\right)^2\right] < \infty$. Thus, the expected loss for one fixed sample can be decomposed:

$$\begin{aligned}
\mathrm{E}\left[\left(y - \hat{f}_{\mathcal{D}_n}(x)\right)^2\right] &\overset{(3.1)}{=} \mathrm{E}\left[\left(f(x) + \varepsilon - \hat{f}_{\mathcal{D}_n}(x)\right)^2\right] \\
&= \mathrm{E}\left[(f(x)+\varepsilon)^2 - 2\left(f(x)+\varepsilon\right)\hat{f}_{\mathcal{D}_n}(x) + \left(\hat{f}_{\mathcal{D}_n}(x)\right)^2\right] \\
&= (f(x)+\varepsilon)^2 - 2\left(f(x)+\varepsilon\right)\mathrm{E}\left[\hat{f}_{\mathcal{D}_n}(x)\right] + \mathrm{E}\left[\left(\hat{f}_{\mathcal{D}_n}(x)\right)^2\right] \\
&\quad + \left(\mathrm{E}\left[\hat{f}_{\mathcal{D}_n}(x)\right]\right)^2 - \left(\mathrm{E}\left[\hat{f}_{\mathcal{D}_n}(x)\right]\right)^2 \\
&= \left(f(x) + \varepsilon - \mathrm{E}\left[\hat{f}_{\mathcal{D}_n}(x)\right]\right)^2 + \mathrm{Var}\left[\hat{f}_{\mathcal{D}_n}(x)\right] \qquad (3.2)
\end{aligned}$$

Here,

$$\begin{aligned}
\mathrm{Var}\left[\hat{f}_{\mathcal{D}_n}(x)\right] &= \mathrm{E}\left[\left(\hat{f}_{\mathcal{D}_n}(x)\right)^2\right] - \left(\mathrm{E}\left[\hat{f}_{\mathcal{D}_n}(x)\right]\right)^2 \\
&= \int_{\mathbb{R}^{2n}} \left(\hat{f}_d(x)\right)^2 dP_{\mathcal{D}_n}(d) - \left(\int_{\mathbb{R}^{2n}} \hat{f}_d(x) dP_{\mathcal{D}_n}(d)\right)^2
\end{aligned}$$

describes the variance of the predicted value for x.
However, as described earlier, to really measure the performance of the predictor, it does not suffice to evaluate the loss function on one sample. Multiple samples in the test set are used to estimate the true expected loss. This can be done by averaging the expected loss dependent on x and ε, which has just been calculated, over $\boldsymbol{X}$ and $\boldsymbol{\mathcal{E}}$. Thus, the final decomposition of the expected loss can be derived:

$$\begin{aligned}
&\mathrm{E}\left[\left(\boldsymbol{Y} - \hat{f}_{\mathcal{D}_n}(\boldsymbol{X})\right)^2\right] \\
&\overset{(3.1)}{=} \int_{\mathbb{R}}\int_{\mathbb{R}}\int_{\mathbb{R}^{2n}} \left(f(x) + \varepsilon - \hat{f}_d(x)\right)^2 dP_{\mathcal{D}_n}(d) dP_{\boldsymbol{\mathcal{E}}}(\varepsilon) dP_{\boldsymbol{X}}(x) \\
&\overset{(3.2)}{=} \int_{\mathbb{R}}\int_{\mathbb{R}} \left(f(x) + \varepsilon - \mathrm{E}\left[\hat{f}_{\mathcal{D}_n}(x)\right]\right)^2 dP_{\boldsymbol{\mathcal{E}}}(\varepsilon) dP_{\boldsymbol{X}}(x) + \int_{\mathbb{R}}\int_{\mathbb{R}} \mathrm{Var}\left[\hat{f}_{\mathcal{D}_n}(x)\right] dP_{\boldsymbol{\mathcal{E}}}(\varepsilon) dP_{\boldsymbol{X}}(x) \\
&= \int_{\mathbb{R}} \left(f(x) - \mathrm{E}\left[\hat{f}_{\mathcal{D}_n}(x)\right]\right)^2 dP_{\boldsymbol{X}}(x) + 2\int_{\mathbb{R}} \left(f(x) - \mathrm{E}\left[\hat{f}_{\mathcal{D}_n}(x)\right]\right) dP_{\boldsymbol{X}}(x) \cdot \underbrace{\int_{\mathbb{R}} \varepsilon dP_{\boldsymbol{\mathcal{E}}}(\varepsilon)}_{=0} \\
&\quad + \int_{\mathbb{R}} \varepsilon^2 dP_{\boldsymbol{\mathcal{E}}}(\varepsilon) - \underbrace{\left(\int_{\mathbb{R}} \varepsilon dP_{\boldsymbol{\mathcal{E}}}(\varepsilon)\right)^2}_{=0} + \int_{\mathbb{R}} \mathrm{Var}\left[\hat{f}_{\mathcal{D}_n}(x)\right] dP_{\boldsymbol{X}}(x) \cdot \underbrace{\int_{\mathbb{R}} dP_{\boldsymbol{\mathcal{E}}}(\varepsilon)}_{=1} \\
&= \underbrace{\int_{\mathbb{R}} \left(f(x) - \mathrm{E}\left[\hat{f}_{\mathcal{D}_n}(x)\right]\right)^2 dP_{\boldsymbol{X}}(x)}_{\text{bias}} + \underbrace{\int_{\mathbb{R}} \mathrm{Var}\left[\hat{f}_{\mathcal{D}_n}(x)\right] dP_{\boldsymbol{X}}(x)}_{\text{variance}} + \underbrace{\mathrm{Var}\left[\boldsymbol{\mathcal{E}}\right]}_{\text{noise}}.
\end{aligned}$$

In the last step, $\mathrm{E}[\boldsymbol{\mathcal{E}}] = 0$ is used.
The first term describes the bias. It is high if the predictor is not close to the underlying function f, for example because the model is not complex enough. It also shows that it is beneficial to have a large dataset to train on, because the expected prediction given x can be estimated more precisely with more data. However, in practice the size of the training set is limited. Therefore, trying to lower this error often leads to overfitted models that rely on a specific training set too heavily.
The second term describes the variance of the predictor. If the predictor is very sensitive to changes in the training dataset, predicted values vary a lot, therefore the variance is high. A possibility to lower this error would be to design a simple model that is underfitted and uninfluenced by the training set. However, this would most likely lead to a model with high bias, as the model does not learn anything.
The third term is the irreducible error. It cannot be lowered by choosing another predictor $\hat{f}_d$. Therefore, in this example, even the best predictor has an expected loss greater than zero. Hence, in order to minimize the expected loss, both bias and variance have to be lowered, although this usually involves a trade-off.
Such a decomposition does not only hold for a regression with the mean squared error as the loss function. Under certain assumptions, other loss functions are possible as well, which makes a similar decomposition applicable for classification problems, see [James, 2003, pp. 124-125].
Overfitting is a big problem when working with financial data. It is common practice to evaluate a financial algorithm by using it on historic data in a walk-forward direction via a backtest, which will be described in Section 5.1. Overly using this method with machine learning algorithms is risky, because the model is essentially fitted on the same dataset over and over again. Given enough trials, the algorithm is likely to make false discoveries, as it can recognize patterns in random noise or irrelevant features. For example, it has been shown that it is possible to find a profitable monthly trading rule on a dataset of random walks with mean 0, although there are no underlying seasonal patterns by design, see [Bailey et al., 2014, p. 468].
There are multiple ways to handle overfitting of machine learning models. When tuning the parameters of the model, it is possible to intentionally reduce the complexity of the model, for example see Subsection 3.3.4. Furthermore, machine learning techniques like cross-validation or ensemble methods can reduce overfitting significantly, which will be discussed in Subsection 3.1.3 and Section 3.4.

3.1.2 Train/Test Split

In order to evaluate a classifier, the labelled dataset is split into two parts: a training and a test subset. The training subset is used by the model to learn patterns to predict class labels. The test subset is completely excluded from this learning stage. When the final model is fitted, it is used on the test subset to derive predictions. As the test set is a subset of the labelled data, the real labels can be compared to the predicted ones, thus allowing a performance measure of the model.
Typically, the samples of the test subset are chosen randomly, so a multitude of different samples can be tested, see [James, 2013, pp. 176-178]. When samples are independent

and identically distributed, that is a valid approach. However, a financial dataset has to be interpreted as a time series.
Suppose there are two samples X_i and X_j that describe the same asset on consecutive days t_i and $t_j = t_i + 1$. Due to serial correlation, the features of X_i at t_i are roughly the same on the next trading day t_j. Most likely, the label y_i of X_i is roughly the same as the label y_j of X_j as well, because it is based on overlapping data. This is a problem when X_i is in the training set and X_j is in the test set. Genuine new and unseen samples do not share this level of similarity of both features and label. To predict the label of X_j, a classifier can thus simply refer to the label of X_i, which is identical most of the time. If that occurs with many samples, the model seems to have spectacular prediction abilities without really learning anything. This process is called leakage, because information of the test set is leaked in the training phase. In these cases, even irrelevant features have some feature importance, which might lead to false discoveries. Most importantly, the performance on new, unseen data is bad, since there are no highly similar labelled samples to derive predictions from. To tackle this, the training and test set should be separated. There are two main ways to do this: purging and embargoing, see [López, 2018, pp. 105-108].

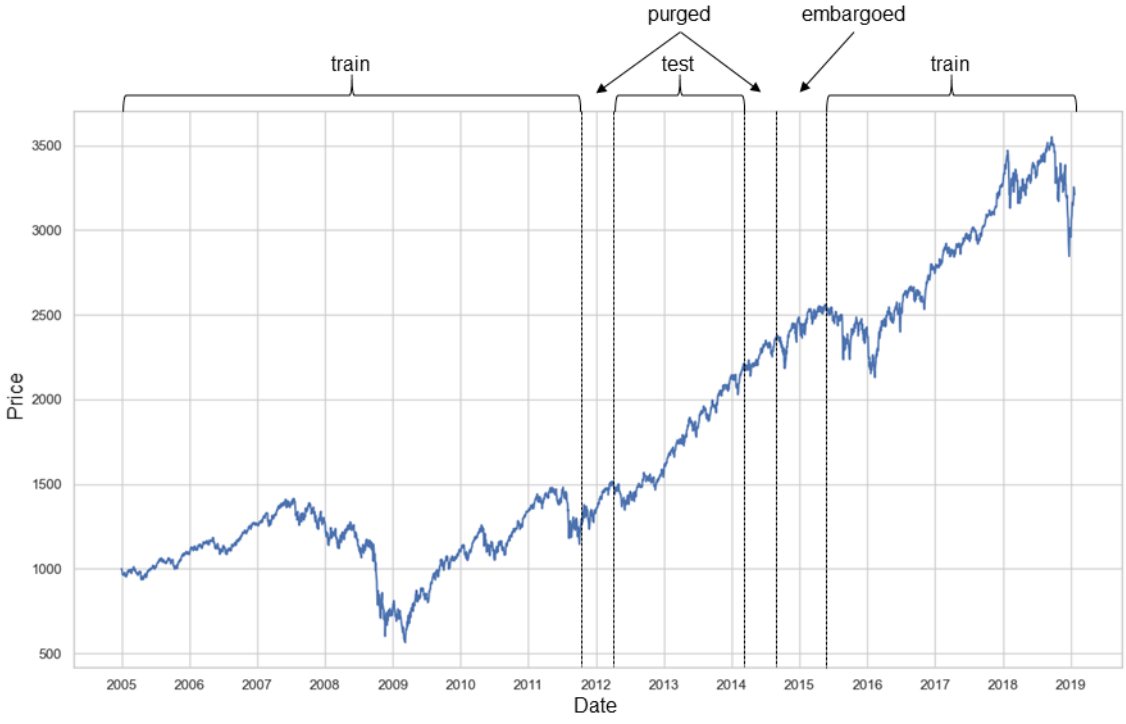

Figure 3.1: The purged and embargoed data visualized in a price chart.

Purging focuses on separating the data used for labelling. As it has been shown in Section 2.2, the label y_i of the sample X_i is derived from the trading days following t_i. When y_i has a horizon of h trading days, the days t_i to $t_i + h$ are affected. Every sample X_j that uses a day from t_i to $t_i + h$ to derive their own label is deleted from the training set, see Figure 3.1. Thus, if samples of the training and test dataset happen to be similar, it is ensured that they do not use the same data to derive their labels.

Purging tries to tackle leakage by deleting neighbouring samples when both features and labels are similar. As long as the labels are independent, it is okay to have similar features. In fact, any classifier actively tries to detect similar feature constellations in order to predict a label, as there might be an underlying relationship. Thus, if features of samples in the training and test set rely on rolling windows, the windows can overlap. However, if the test set precedes the training dataset, the features of the training set could have access to information in the test set. Deleting the first samples of these training sets is called embargoing. There is no need to embargo samples if the training set precedes the test set, as all information of the training samples prior to the test is available at the testing time.
Now suppose there is a dataset that ranges from 1990 to 2019, as there is in Chapter 6. A very simple way to evaluate a model would be to fit it on the first part of the data, for example on the first 80%, and to test it on the rest in order to minimize the need to embargo data. However, that way the model would be tested on recent samples only. For example, in the 80% split, the data from 2013 to 2019 would be used for testing. These years showed a relatively steady growth. If models would be tested on these years only, models that predict an upward price movement are favoured. When a financial crash occurs, like in the beginning of 2020, the model is not prepared. In the training set, it has access to data of past crashes, but the performance in a crash situation has never been evaluated, so models that might perform well could have been discarded.
A better way to use the full potential of a training set is cross-validation, where the model is trained multiple times on different parts of the dataset.

3.1.3 Cross-Validation

In k-fold cross-validation, the dataset is divided into k subsets of roughly the same size. One of these subsets is used for testing, the others are used to train the model. When a financial dataset is used, this means that a test set can precede a training set. This process is repeated k times, so every subset serves as a test set exactly once, as seen in Figure 3.2.

Figure 3.2: 5-fold cross validation.

For every fold, the performance is noted, more on that in Subsection 3.1.4. The values of every fold are averaged to derive a final score. Furthermore, the standard deviation of these values can be calculated. A high standard deviation implies a high variance of the model. A model with a high score and a low standard deviation is desirable. For more details, see [James, 2013, pp. 181-186].
In the context of financial data, when the dataset is indexed chronologically, every subset spans a fixed time period in order to avoid leakage. If the number of observed assets drastically changes over time, the sizes of the subsets might need to be altered. In every fold, purging has to be applied. If the test set precedes the training set, there should be an embargo as described earlier.
The cross-validation method can be used to evaluate the final model. More importantly, cross-validation is used to adjust the parameters of a model in beforehand, which is called hyperparameter tuning. It is possible to test a parameter configuration of a model on the whole set and pick the best parameters. However, this is not a good practice, as the test set is used multiple times, which might lead to overfitting, see [Raschka & Mirjalili, 2015, pp. 296-297]. It is better to only apply the cross-validation method on the training set. The training set is thus divided into a smaller training and a validation set multiple times. In the end, the model with the best performance can be tested on the real test set, giving a reliable evaluation of its performance on new, unseen data. This scheme is used later in Chapter 6.

3.1.4 Scoring

A supervised learning classifier tries to predict the class membership of samples. How to choose the right labelling method to build classes has been discussed in Section 2.2. Now, the question is how to evaluate the performance of the classifier on labelled data. There are different ways to measure the performance. Each measure has advantages and drawbacks, which will be discussed in the following.

Accuracy

A classifier has to predict the label y_i of the sample X_i for $i \in D_{\text{test}} \subseteq \{1, \ldots, N\}$, where N denotes the number of samples in the dataset. An easy method to check these predictions is to simply give the proportion of correct predictions. If $\hat{y}_i$ denotes a predicted label, the accuracy can be expressed as

$$\text{accuracy} = \frac{1}{|D_{\text{test}}|} \sum_{i \in D_{\text{test}}} \mathbb{1}_{y_i}(\hat{y}_i) \in [0, 1],$$

where

$$\mathbb{1}_{y_i}(y) = \begin{cases} 0 & \text{if } y_i \neq y \\ 1 & \text{if } y_i = y \end{cases}$$

The accuracy is a very intuitive measure. However, when the labels are not equally distributed, for example when there are 70% positive and 30% negative labels, relying on the accuracy can be problematic. If the model were to always predict a positive label, it would have an accuracy of 70% without really learning anything.

To circumvent this, it is a good idea to check a visualization of the true and the predicted labels. In a confusion matrix, as seen in Figure 3.3, there are four possibilities. The true label is either -1 or 1 and the predicted label is either -1 or 1. By categorizing the samples of the test set, it is possible to see the difference of accuracy in between classes. For example, the classifier depicted in Figure 3.3 mostly predicts positive labels. Unfortunately, the majority of negatively labelled samples are predicted to be positive as well. If the label distribution is skewed, the accuracy might not take this into account.
However, in some situations like hyperparameter tuning, the program cannot check a visualization like the confusion matrix. It has to rely on a single number to evaluate the model's performance. Therefore, there are other measures that integrate the confusion matrix.

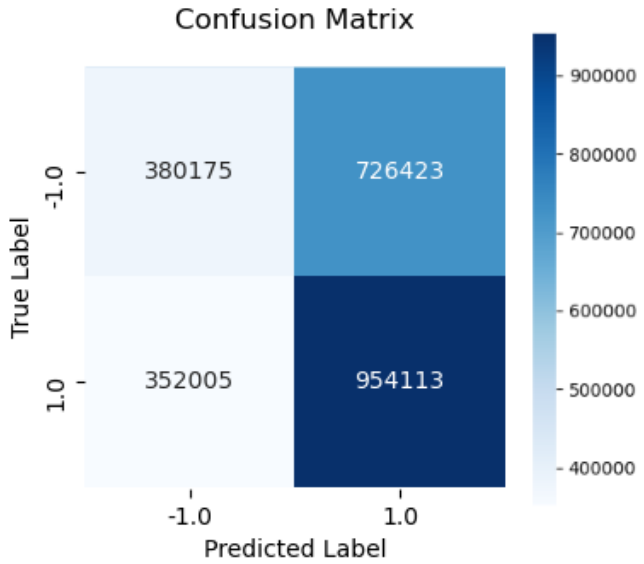

Figure 3.3: Depending on the true and predicted label of a sample, a confusion matrix depicts the number of samples that belong to one of the four categories.

F_1 Score

Another way to visualize the confusion matrix is seen in Figure 3.4. The square, which stands for a set of observations, is divided into two rectangles. The left half represents samples that are labelled positively, the right half represents negatively labelled samples. A binary classifier tries to predict the label of observations. This is represented by the circle, as all samples in the circle are predicted to have a positive label and vice versa. There are essentially 4 different possibilities:

1. TP (true positive): The model correctly identifies a positive label.
2. FP (false positives): The model wrongly assigns a positive label to a negative labelled sample.
3. TN (true negative): The model correctly identifies a negative label.

4. FN (false negative): The model wrongly assigns a negative label to a positive labelled sample.

The goal of every classifier is to correctly predict all labels, in essence to maximize the TP and TN area. However, minimizing the FP area generally increases the FN area, as fewer positive labels are predicted. For more details, see [López, 2018, p. 52].

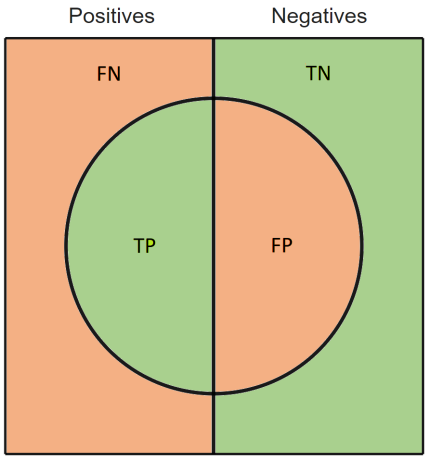

Figure 3.4: A visualisation of possible errors

Using this terminology, one is able to calculate the accuracy by

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \in [0, 1].$$

In the context of finance, it is important not to wrongly predict an upward price movement, which can be costly when an investment is conducted. This aspect is modelled by the precision, which can be expressed as

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \in [0, 1].$$

Missing out on positive price movements can be problematic as well. This is modelled by the recall, described by

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \in [0, 1].$$

It is desirable to have both a high precision and a high recall. Both properties can be combined in a single score, the F_1 score, by calculating the harmonic mean:

$$\text{F}_1 = \left(\frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \right) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \in [0, 1].$$

The F_1 score deals with many problems of the accuracy, as it balances both precision and recall. If the classifier always predicts a negative label, the recall is 0, thus the F_1 score is 0 as well. However, if the majority of labels are positive, a classifier always predicting a positive label has precision = accuracy and recall = 1, therefore the F_1 score is high as well. In that case, a solution would be to switch the definition of positive and negative labels, as shown in [López, 2018, p. 206].

Negative log Loss

In the financial context, a key component missed by both accuracy and F_1 score is the model's certainty in its predictions. When the model predicts an upward price movement of an asset with high probability, an investor might be inclined to buy that asset. If the prediction is wrong, a loss is realized. Contrarily, if the predicted probability of a positive label is slightly above 50%, the prediction is not really useful. When classifying in two classes, predictions with low certainty function as a third "neutral" class, where the exact direction of the price movement is ambiguous.
To take the model's confidence into account, a measure can be based on the log loss of the probability of a right prediction, see [López, 2018, pp. 134-135]. For a feature matrix $X \in \mathbb{R}^{N \times M}$, let $y \in \{-1, 1\}^N$ denote the true labels and $p \in [0, 1]^N$ the predicted probabilities that a sample belongs to the positive class. Thus, the negative log loss for the test set $D_{\text{test}} \subseteq \{1, \dots, N\}$ can be estimated by

$$L(y, p) = \frac{1}{|D_{\text{test}}|} \sum_{i \in D_{\text{test}}} \left(\mathbb{1}_1(y_i) \log(p_i) + \mathbb{1}_{-1}(y_i) \log(1 - p_i) \right) \leq 0,$$

where $\mathbb{1}_.$ denotes the indicator function. Both the accuracy and the F_1 score assign high values for good performance. To be consistent, the negative log loss is favoured over the positive log loss. This way, high values represent a good score.
For example, suppose the model falsely predicts a positive label $\hat{y}_i = 1$ for a sample X_i. The model also reports its confidence in this decision with the value $p_i \in [0.5, 1]$. The value can be interpreted as the predicted probability that the sample belongs to the positive class. In this example, it is bigger than 50%, because the classifier predicts a positive label. If this confidence had been lower than 50%, a negative label would have been predicted. As the prediction is wrong, the opposite value $1 - p_i$ is used in the negative log loss. The higher the false predicted probability p_i is, the lower $\log(1 - p_i)$, so over-confident wrong predictions have a much lower negative log loss than uncertain predictions. Similarly, correct predictions with high confidence are preferred over predictions with low confidence.
For the reasons listed above, in this thesis the negative log loss is used as the main score to measure the performance of the models while hyperparameter tuning. However, given class unbalances of the dataset, this measure is adjusted by using sample weights, which is explained in the next section.

3.2 Sample Weights

If there is a big class unbalance, it might be beneficial to give the samples of the underrepresented class more weight, to ensure that both classes are equally important. This can be beneficial when a model is trained, see Subsection 3.3.3. Furthermore, sample weights can be used when evaluating a model to ensure that samples of minor classes are not overlooked.
If the sample weight is calculated with the label, it has to be derived with the training set only. The sample weights for the test set can be applied afterwards. For example, suppose a class is underrepresented in the training set. If samples that belong to this

class are given a high weight w, the same weight w is then used on samples of the test set that also belong to this class. Of course, in this case, the model is not allowed to see the sample weights of the test set before giving its predictions, as information about the true label would be leaked. Rather, the sample weights can be applied afterwards to evaluate the model.
Typically, the weights are chosen inversely proportional to class frequencies. Let $D_{\text{train}} \subseteq \{1, \dots, N\}$ denote a training set. Furthermore, let C denote the number of classes and N^c_{train} the number of samples in D_{train} that belong to class c. Then, the average number of samples in each class is calculated by

$$\frac{1}{C} |D_{\text{train}}| = \frac{1}{C} \sum_{c=1}^{C} N^c_{\text{train}}.$$

The weight inversely proportional to class frequencies can be calculated by

$$f(c, D_{\text{train}}) = \frac{1}{N^c_{\text{train}}} \frac{1}{C} \sum_{c'=1}^{C} N^{c'}_{\text{train}}.$$

This is used to set the weight for a sample $i \in \{1, \dots, N\}$ that belongs to class c:

$$w_i = f(c, D_{\text{train}}).$$

This way, class unbalances are compensated. Of course, the training set has to be large enough to contain every possible class multiple times. If a class has no related sample in the training set, a classifier cannot predict that particular class. Accordingly, the sample weight would be set to zero.
In theory, other weighting schemes are possible. For example, it might be beneficial to give extraordinary samples of sudden shocks of the stock market more weight. For more on this, see [López, 2018, pp. 59-72].
To use sample weights when evaluating the performance of a model, the scoring measures have to be adjusted. Exemplary, this is done for the negative log loss.
For a test set $D_{\text{test}} \subseteq \{1, \dots, N\}$ and a weight vector $w \in \mathbb{R}^N$, let

$$\widetilde{N}_{\text{test}} = \sum_{i \in D_{\text{test}}} w_i$$

denote the weighted number of samples in the test set. Using the same notation as in Subsection 3.1.4, the weighted negative log loss can thus be defined by

$$\widetilde{L}(y, p, w) = \frac{1}{\widetilde{N}_{\text{test}}} \sum_{i \in D_{\text{test}}} w_i \Big(\mathbb{1}_1(y_i) \log(p_i) + \mathbb{1}_{-1}(y_i) \log(1 - p_i) \Big) \leq 0.$$

3.3 Decision Tree

After explaining model evaluation methods, this section is dedicated to explain the actual model used to predict stock price movement. As explained earlier, the models are supervised learning classifiers, using labelled data to learn patterns that help to

predict the label on new, unseen data. There are many different classifiers that could be used in finance. This thesis focuses on a simple model, a decision tree, and shows how ensemble methods improve predictions. In theory, the base model can be exchanged with other classifiers, for example neural networks.

3.3.1 Tree Structure

A decision tree is an intuitive tool to structure a complex decision making process. As this structure comes naturally, it is often applied unconsciously. For example, a decision tree can be used to decide whether to buy a certain stock, as seen in Figure 3.5. For the strategy depicted, see [Giese, 2015, p. 225].

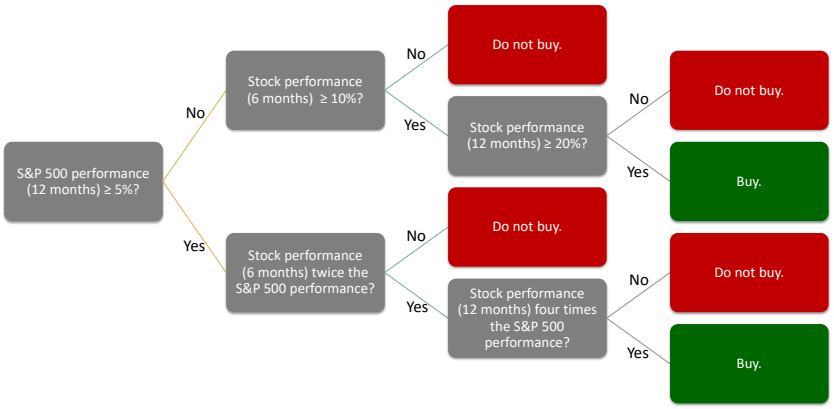

Figure 3.5: A decision tree of a trading strategy.

The decision tree can be understood as a series of questions, leading to a decision. The tree consists of nodes, in which these questions are formulated. There is a single root node, which is followed by multiple child nodes. A node can have multiple child nodes. For computational reasons, the decision trees used in this thesis are binary, meaning that every node has two child nodes, although more nodes are possible. There can be multiple levels of internal nodes. At the end of every branch, there are the so called leaves, which make a decision.
The nodes and decisions in Figure 3.5 are chosen by hand, using years of trading experience. A computer can learn from the data and derive a similar structure by itself. It chooses the features that split the dataset and decides on the optimal threshold for every split. Such a decision tree can be seen in Figure 3.6. The classifier is build on a dataset of Apple Inc., ranging from 1990 to 2004, and tries to predict the Edge Ratio label with a horizon of 120 trading days, as described in Subsection 2.2.3.
Starting at the root node, the algorithm recursively splits the dataset on the feature that results in the largest information gain, which will be defined later. One child node consists of the part of the dataset that possesses a certain characteristic, the other part does not. In Figure 3.6, the root node is split on the feature 'ReturnMrktRatio250', which describes the ratio of the return of Apple Inc. and the S&P 500 return. The returns are

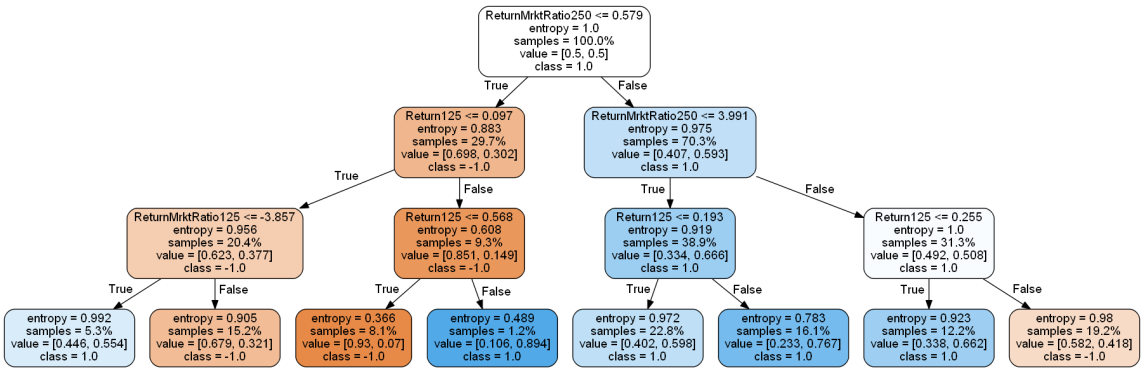

Figure 3.6: A decision tree classifier for the stock price movement of Apple Inc.

based on the previous 250 trading days, which is roughly a year. As the return can turn negative, this ratio can also turn negative if the stock price of Apple Inc. runs opposite to the market. This is problematic if the S&P 500 return switches its sign, because a value near zero results in a big ratio that then suddenly switches its sign. Clearly, this feature has to be improved, more on that in Chapter 4. However, for the sake of simplicity, 'ReturnMrktRatio250' just describes the unaltered ratio in this example.
For 29.7% of the samples in the training set, this ratio is smaller than 0.579. For this part of the dataset, the distribution of the classes has changed. While the positive and negative Edge Ratio labels are distributed evenly in the whole dataset (indicated by 'value'), after the first split 69.8% of the samples in the left dataset belong to the negative class, which means there is a negative outlook for the price movement. Using this subset of samples, the dataset is split on another feature, the 125 day return. If the return is bigger than 9.7%, the proportion of negative labelled samples rises to 85.1%. Finally, the subset in the node is split for a last time, again using the 125 day return. This split proves very efficient, as the proportion of negative labels in the left child node rises to 93%, while the proportion of positive labels in the right child node is 89.4%. These nodes have no child nodes, so they are called leaves.
When there is a new observation, for which the label should be predicted, the decision tree follows the questions in the nodes until it arrives at a leaf node. There, samples possess similar features to the observation. The final prediction of the label is decided by the majority of labels that have been observed in this part of the dataset. In Figure 3.6, a sample having a market return ratio smaller than 0.579 and a 125 day return greater than 56.8% is predicted to belong to the positive class with a probability of 89.4%, following the class distribution of the dataset in the leaf node.
Decisions made by a decision tree are very transparent and easy to understand, which is a big advantage to other classifiers. Furthermore, decision trees can give a great insight in the dataset. For example, studying Figure 3.6, it is apparent that having a market return ratio greater than 3.991 has almost no effect on the Edge Ratio label in this particular dataset, as the class distribution stays nearly even (49.2% to 50.8%). This information can be used to improve trading strategies.

3.3.2 Information Gain

At every node, the algorithm is supposed to split the dataset on the most characteristic feature. Ideally, a leaf node is pure, which means that every sample of the subset belongs to the same class. In the process of developing a tree, splits that further an uneven class distribution are favoured. An uneven class distribution can be quantified with an impurity measure. Most commonly, there are two measures used for decision trees: the entropy and the Gini impurity, see [Raschka & Mirjalili, 2015, pp. 157-164].
For the dataset D_t at a node t, the proportion of samples that belong to class $c \in \{1, \dots, C\}$ can be defined by

$$p(c|t) = \frac{1}{N_t} \sum_{i \in D_t} \mathbb{1}_{y_c}(y_i),$$

with N_t as the number of samples in D_t, y_i as the label of a sample i and y_c as the label of class c. Obviously, for every node t:

$$\sum_{c=1}^{C} p(c|t) = 1.$$

The entropy can be defined as

$$I_H(t) = -\sum_{c=1}^{C} p(c|t) \log_2 p(c|t).$$

Since $p(c|t) \in [0, 1]$, $\log_2 p(c|t) \leq 0$ and therefore $I_H(t) \geq 0$. Using l'Hôpital's rule, it is easy to show that $p \log_2 p \to 0$ for $p \to 0$. The entropy is zero when all samples of a set belong to the same class. The entropy is maximal when all classes have the same amount of samples, which can be shown using Lagrange multipliers.
Using the same notation, the Gini impurity can be defined as

$$I_G(t) = \sum_{c=1}^{C} p(c|t)(1 - p(c|t)).$$

Similar to the entropy, it is maximized when class sizes are equal.
As seen in Figure 3.7, both measures are similar, which leads to similar results in practice, see [Raileanu & Stoffel, 2004, p. 92]. Calculating the Gini impurity is faster, but only slightly.
Having a measure for the class impurity $I \in \{I_H, I_G\}$, it is now possible to define the information gain (IG) at every split for a feature f:

$$\mathrm{IG}(t_p, f) = I(t_p) - \frac{N_l}{N_p} I(t_l) - \frac{N_r}{N_p} I(t_r),$$

where t_p denotes the parent node and N_p describes the number of samples at the parent node. Similarly, t_l and t_r describe the left and the right child node and N_l and N_r the number of samples at the respective nodes. Hence, the information gain can be summarized as the weighted increase in impurity, if a node is split on a certain feature.

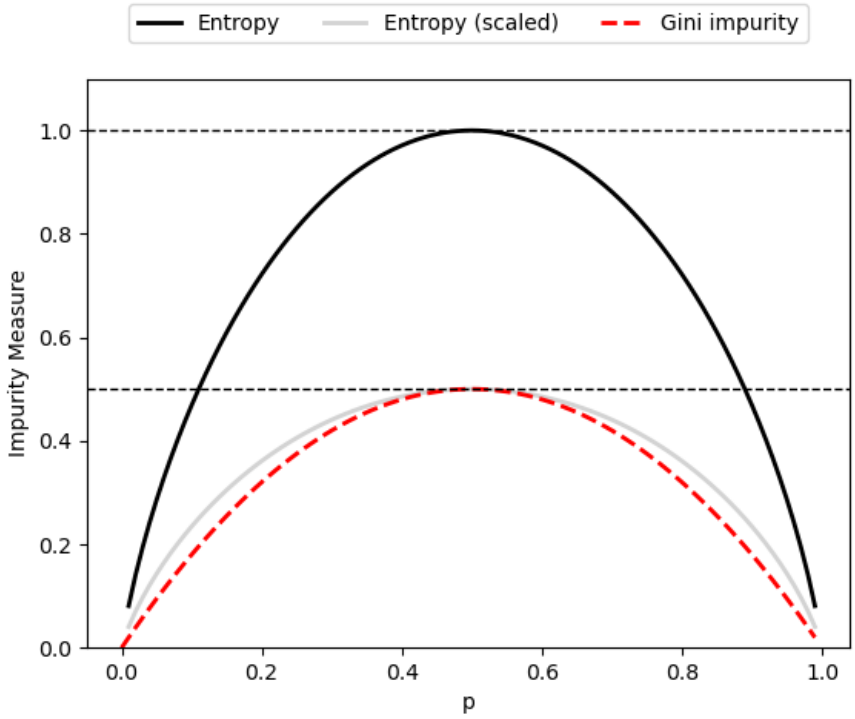

Figure 3.7: Entropy and Gini impurity for two classes.

Weighting the impurity measure of every child node favours big splits.
In order to calculate the information gain for a feature, a threshold to split the dataset on has to be set. To derive the optimal threshold for a feature, the samples in the dataset of a node are sorted in regards to that feature. Let N_t denote the number of samples in the dataset of the node t. As this number is finite, it suffices to test $N_t - 1$ values for the threshold to derive every possible split. The threshold with the biggest information gain is chosen as the value to split the dataset on, see [Quinlan, 1993, pp. 25-26].
In the process of building a decision tree, the feature that yields the biggest information gain is chosen at every split. The algorithm ends when splitting the remaining nodes yields no information gain, typically when the nodes are pure.

3.3.3 Weighted Information Gain

As described earlier, to predict the class of a new sample, the algorithm follows the nodes, until it arrives at a leaf where the subset of the data is similar to the new sample. Then, the predicted label is simply based on the majority of samples in this leaf. If there is a great class unbalance, it is possible that small classes are overshadowed by the predominant class, thus preventing the prediction of the minor class. To address this, it is possible to set sample weights, which are taken into account at every split and in the final prediction, see Section 3.2.
In order to use sample weights w_i for $i \in \{1, \dots, N\}$, a few changes have to be made. Instead of using the number of samples in the dataset of a node t, the weighted number

of samples

$$\widetilde{N}_t = \sum_{i \in D_t} w_i$$

is utilized. Using the same notation as before, the proportion of samples that belong to a class c at the node t has to be updated:

$$\widetilde{p}(c|t) = \frac{1}{\widetilde{N}_t} \sum_{i \in D_t} \mathbb{1}_{y_c}(y_i) w_i.$$

Rather than relying on the simple majority of samples, this weighted proportion is used in the leaves to derive the final label prediction of new, unseen samples.
Furthermore, weighting has influence on the building process of the tree as well. In the definition of the impurity measures, the weighted proportion $\widetilde{p}$ is used instead of the normal proportion p. To make a distinction to the previous definitions, these weighted impurity measures will be referred to as $\widetilde{I}_H$ and $\widetilde{I}_G$. Finally, the weighted information gain can be defined by

$$\widetilde{IG}(t_p, f) = \widetilde{I}(t_p) - \frac{\widetilde{N}_l}{\widetilde{N}_p} \widetilde{I}(t_l) - \frac{\widetilde{N}_r}{\widetilde{N}_p} \widetilde{I}(t_r).$$

Obviously, the old definition of the information gain is a special case of the weighted information gain for uniform sample weights $w_i = 1$ for all $i \in \{1, \ldots, N\}$.

3.3.4 Regularization

The basic decision tree algorithm tries to split the dataset until every leaf is pure. When the classification problem is complex, the borders that separate classes become fuzzy. A split on a node is rarely perfect, as there often remain samples that possess similar features but have different labels. In that case, the basic algorithm keeps splitting the dataset until the leaves consist of one sample only. This is a problem, because a new, unseen sample that happens to be similar to the sample in the one-sample leaf is predicted to belong to the same class with a predicted probability of 100%, giving a false sense of certainty. In practice, these kind of decision trees are extremely overfitted, they might give good predictions on the training set, as samples are alike, but they struggle to give reliable predictions on new, unseen data. Therefore, in order to tackle overfitting, regularization is mandatory.
There are different regularization methods, depending on the machine learning model. For decision trees, the easiest way is to set a maximum depth. This way, after the dataset has been split a predefined number of times, the algorithm stops. The leaves are not pure, but this is reflected in the low predicted probability that a new sample belongs to a certain class. When sample weights are used, rather than setting a maximum depth, it is a good idea to set the minimum sum of sample weights a leaf is allowed to consist of. That way, class unbalances can be considered. Another way to stop the development of the decision tree early is to set a minimum information gain that is required to split a node.
These are a few examples of so called hyperparameters that have to be defined before

fitting the machine learning model. Unfortunately, there is rarely a perfect choice, because there is usually a trade-off between bias and variance, as described in Subsection 3.1.1. For example, if the maximum depth of a decision tree is too high, the model is overfitted and does not generalize well. If, however, the maximum depth is set too low, the model is underfit and the tree is too simple.
Via cross-validation, every possible combination of these so called hyperparameters is tested on the training set. The combination of parameters that yields the best performance is then used in the final model.
Unfortunately, even when regularization is used extensively, in practice the performance of decision trees is underwhelming. A much better strategy is to use decision trees as the building block of more complex machine learning models. Rather than relying on a single decision tree, these models combine the predictions of an ensemble of trees to derive a much better prediction.

3.4 Ensemble Methods

In machine learning, it is common to combine the predictions of multiple classifiers to improve the overall performance. There are several ways to structure such an ensemble method. The easiest way is to fit multiple different classifiers and to predict the label that has been predicted by the majority of these classifiers. This approach is a simplification of the so called bootstrap aggregating method, which will be discussed in Subsection 3.4.1. This method often performs better than the best classifier of the ensemble on its own. When a variation of bootstrap aggregating is used on decision trees, one of the most popular machine learning models is formed, the so called random forest. This model is discussed in Subsection 3.4.2.
Apart from bootstrap aggregating, there are other forms of ensemble methods. Instead of fitting several classifiers independently in parallel, boosting fits them sequentially. Every time a classifier is fit, the next classifier is adjusted so that it avoids the prior classifier's mistakes. This method often performs very well. However, boosting is mainly focused on improving the bias, which often comes at the expense of a higher variance, as explained in Subsection 3.1.1. As overfitting is a big problem on financial data, boosting often performs worse than bootstrap aggregating, which mainly improves the variance, see [López, 2018, pp. 100-101]. That is why the main focus of this thesis is bootstrap aggregating.
Of course, ensemble methods are not only applicable on classification problems. All of the ensemble methods that are introduced in this thesis can be easily adjusted to predict continuous values in a regression problem as well.

3.4.1 Bootstrap Aggregating

Suppose there are two classes $\{-1, 1\}$ and n different classifiers

$$\hat{f}_{\cdot}^{(j)} : \mathbb{R}^{N\times(M+1)} \times \mathbb{R}^{M} \to \{-1, 1\}, \quad (d, x) \mapsto \hat{f}_{d}^{(j)}(x) = y^{(j)},$$

for $j \in \{1, \dots, n\}$. Basically, the classifier is fitted on a training dataset $d \in \mathbb{R}^{N\times(M+1)}$, which is a feature matrix with N samples and corresponding labels, to produce a pre-

diction on an unlabelled sample $x \in \mathbb{R}^M$. Similar to Subsection 3.1.1, the training dataset d can be interpreted as a realization of a random matrix $\boldsymbol{\mathcal{D}}$ on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$.
Suppose a realization d of $\boldsymbol{\mathcal{D}}$ is fixed. To diversify the classifiers, every classifier $\hat{f}_\cdot^{(j)}$ is fitted on a slightly different training dataset $d_b^{(j)}$. This is achieved by randomly choosing N samples in the original training dataset d with replacement. To describe this process, which is called bootstrapping, define a uniformly distributed random vector

$$\boldsymbol{B} : (\Omega, \mathcal{F}) \to \left(\{1, \ldots, N\}^N, \mathcal{P}\left(\{1, \ldots, N\}^N\right)\right), \quad \omega \mapsto \boldsymbol{B}(\omega)$$

with $P_{\boldsymbol{B}} \sim \mathcal{U}(\{1, \ldots, N\}^N)$. In essence, $\boldsymbol{B}$ describes N random drawings with replacement out of the index set $\{1, \ldots, N\}$. Thus, only about 63% of the original samples are used in an average bootstrapped dataset, see [Géron, 2019, p. 195].
Based on a dataset $d \in \mathbb{R}^{N \times (M+1)}$, bootstrapping can thus be defined as

$$d_{\boldsymbol{B}} : (\Omega, \mathcal{F}) \to \left(\mathbb{R}^{N \times (M+1)}, \mathcal{B}\left(\mathbb{R}^{N \times (M+1)}\right)\right), \quad \omega \mapsto d_b = (d_{i,\cdot})_{i \in \boldsymbol{B}(\omega)}.$$

This bootstrap process is performed for every classifier $\hat{f}_\cdot^{(j)}$, so multiple datasets $d_b^{(j)} = d_{\boldsymbol{B}}(\omega_j)$ are constructed for $\omega_j \in \Omega$ with $j \in \{1, \ldots, n\}$. The classifiers are then fitted on their individual dataset and their predictions are aggregated. In other words, every predictor votes on what label it would predict. The final prediction is based on the majority of these votes:

$$\hat{f}_\cdot : \mathbb{R}^{N \times (M+1)} \times \mathbb{R}^M \to \{-1, 1\}, \quad (d, x) \mapsto \hat{f}_d(x) = \operatorname{sgn}\left(\sum_{j=1}^{n} \hat{f}_{d_b^{(j)}}^{(j)}(x)\right) = y,$$

where

$$\operatorname{sgn}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases}$$

denotes the sign function. This method is called hard voting.
Another more sophisticated method uses the fact that most classifiers predict the probability $p \in [0, 1]$ that a sample belongs to the positive class, for example as explained in Subsection 3.3.1. Using the same notation as before, the individual classifiers can also be defined as

$$\hat{g}_\cdot^{(j)} : \mathbb{R}^{N \times (M+1)} \times \mathbb{R}^M \to [0, 1], \quad (d, x) \mapsto \hat{g}_d^{(j)}(x) = p^{(j)},$$

for $j \in \{1, \ldots, n\}$. The predicted label $y^{(j)}$ is 1, iff $p^{(j)} > \frac{1}{2}$.
Instead of aggregating the predicted labels of the individual classifiers, the predicted probabilities can be averaged:

$$\hat{g}_\cdot : \mathbb{R}^{N \times (M+1)} \times \mathbb{R}^M \to [0, 1], \quad (d, x) \mapsto \hat{g}_d(x) = \frac{1}{n} \sum_{i=1}^{n} \hat{g}_{d_b^{(j)}}^{(j)}(x) = p.$$

Again, the predicted label is 1 iff $p > \frac{1}{2}$. Hence, predictions of classifiers that have a high value for $p^{(j)}$ and thus a high confidence in their prediction are given more weight.

This method is called soft voting.
Bootstrap aggregating can greatly reduce the variance of the prediction, especially when the classifiers in the ensemble are diversified. Furthermore, the bias can be improved as well. For a detailed discussion, see [López, 2018, pp. 94-97].

3.4.2 Random Forest

Similarly to the bootstrap aggregating method, a random forest combines multiple decision trees by the soft voting method. For that purpose, the underlying dataset is again bootstrapped to fit every decision tree on a slightly different dataset.
Additionally, the random forest algorithm optimizes the bootstrap aggregating method on decision trees by introducing another layer of randomness. As described in Section 3.3, a decision tree is developed by splitting the training dataset multiple times on the most characteristic feature, respectively. To further diversify the decision trees in the ensemble, only a random subset of features is evaluated at every split. Thus, the bias of an individual decision tree might increase, as it perhaps cannot rely on its most characteristic features at every split. However, this also leads to very diversified classifiers in the ensemble, which in turn decreases both variance and bias of the overall random forest.
The number of decision trees in the ensemble and the size of the random subset of samples are hyperparameters of the random forest that have to be chosen in beforehand. Together with the hyperparameters of the decision tree, the optimal combination of hyperparameters in regards to the negative log loss as described in Subsection 3.1.4 is determined via cross-validation on the training dataset.

Chapter 4

Feature Importance

As explained in Section 2.1, samples of financial data are characterized by multiple features. For example, possible features of a sample are the close price or the trading volume at the sample's trading day. If meaningful features are added to a classifier, its predictions become better. Likewise, if redundant or insignificant features are removed, the variance of a model usually decreases and its performance is improved. To identify helpful features, the model's performance when certain features are added or removed from the model can be monitored. Another possibility is to use the feature importance, which will be defined in Subsection 4.1.
Apart from improving machine learning models, the feature importance can be used as a tool to improve trading strategies in general, which will be discussed in Section 4.2.

4.1 Mean Decrease Impurity

As the classifiers used in this thesis are based on decision trees, a good choice for the feature importance is the mean decrease impurity. Unfortunately, it is not applicable to classifiers that are not based on decision trees.
As explained in Section 2.1, financial datasets can be summarized in the feature matrix

$$X \in \mathbb{R}^{N \times M}.$$

For $i \in \{1, \dots, N\}$, a sample $X_i \in \mathbb{R}^{1 \times M}$ describes a financial instrument on a specific trading day. The columns of X describe characteristics of that sample, for example its close price. The feature importance indicates how much information that particular feature possesses. If the feature that has been constructed helps to improve predictions, chances are it really contains valuable information that can be used to construct a better trading strategy.
The mean decrease impurity (MDI) uses the fact that a decision tree is built by evaluating the information gain at every split, see Subsection 3.3.2. In the following, it is defined for a random forest. The MDI for a decision tree can be derived by simply setting the number of trees in the random forest to 1.
For a random forest with n decision trees $T_1, \dots, T_n$, let $T \in \{T_1, \dots, T_n\}$ denote a decision tree. Every decision tree consists of multiple nodes, which are denoted by $t \in T$. The number of samples at a node t is denoted by $N_t \in \{1, \dots, N\}$. If a node t is not a

leaf node, the dataset of the node is split. The feature f a node is split on is denoted by $s(t) = f$. The mean decrease impurity for a feature f can be defined by

$$\mathrm{MDI}(f) = \frac{1}{n} \sum_{T} \sum_{\substack{t \in T: \\ s(t)=f}} \frac{N_t}{N} \mathrm{IG}(t, f),$$

where $\mathrm{IG}(t, f)$ denotes the information gain of feature f at node t as defined in Subsection 3.3.2. In essence, the MDI thus describes the average information gain a feature contributes.
In order to better display the mean decrease impurity for a feature f, it is normalized:

$$\mathrm{MDI}^*(f) = \mathrm{MDI}(f) \left(\sum_{f' \in F} \mathrm{MDI}(f') \right)^{-1} \in [0, 1],$$

where F denotes the set of features. This normalized mean decrease impurity is used as the feature importance in the rest of this thesis.
In the process of building a decision tree, the training dataset is split several times on the feature that respectively yields the biggest information gain. At every split, the number of possible features the dataset can be split on should be limited to 1. Thus, the algorithm simply chooses one feature randomly. This ensures that weak features are not overshadowed by strong ones, because otherwise the decision tree would always choose the feature that yields the biggest information gain, thus inflating the importance of good features. By limiting the number of features to choose from at every split to 1, every feature has the same chance to contribute in decreasing the impurity of the dataset.
It also should be noted that features that have a big range of values generally have a higher MDI, as a significant information gain can be achieved in multiple splits. There are other ways to define the feature importance, each of them with advantages and disadvantages. For a detailed discussion of different feature importances, see [López, 2018, pp. 113-127].

4.2 Improving Trading Strategies

When a new trading strategy is developed, it has to be tested somehow. Usually, this is carried out by using a walk-forward backtest, see Section 5.1. In essence, this method uses a historic dataset to simulate how the trading strategy would have performed if it were carried out in the past. The idea is that a trading strategy that performed well in the past is likely to do so in the future as well. This testing scheme is perfectly valid. However, problems might arise if a trading strategy is backtested on the same dataset over and over again, for example to fine-tune certain parameters. As it has been explained in Subsection 3.1.1, this might lead to a trading strategy that is overfitted to the underlying dataset.
An example for a potentially overfitted trading strategy is the '16-weeks-strategy' by Thomas Gebert. It advises to only buy and sell in certain weeks in a repeating cycle of 16 weeks, whereby the start of this cycle can be chosen arbitrarily, see [Gebert, 2020].

Although there is no explanation why these weeks are chosen in particular, this strategy performs very well on the DAX in a walk-forward backtest. However, it is debatable whether such a strategy really has detected a meaningful pattern in the financial market or whether buying in week 13, 14 and 15 and selling in week 8, 11 and 16 just happens to be the optimal combination of weeks on the underlying historic dataset. In the latter case, there is no guarantee that the strategy performs well on new data. Gebert even admits that usually, when a trading strategy is detected, it does not work on new data, as its 'peak performance' was reached. This might indicate that many of the trading strategies he detects are overfitted. Rather than giving a genuine insight in the inner workings of the financial market, overfitted trading strategies detect random patterns in the training dataset and attribute them meaning.
To investigate whether the financial market really follows the cycle of 16 weeks, the strategy can be translated into a feature and added to a feature matrix. Let the feature '16-weeks-cycle' describe the position of a sample in this cycle, with values ranging from 1 to 16 to indicate the week a financial instrument is located in. The feature matrix is then used to predict the label of the samples as described in Section 2.2. Of course, the feature '16-weeks-cycle' is only a simplification of the whole trading strategy. Nevertheless, a high feature importance would indicate a genuine discovery. Unfortunately, Figure 4.1 shows that the feature on its own does not contain much information.

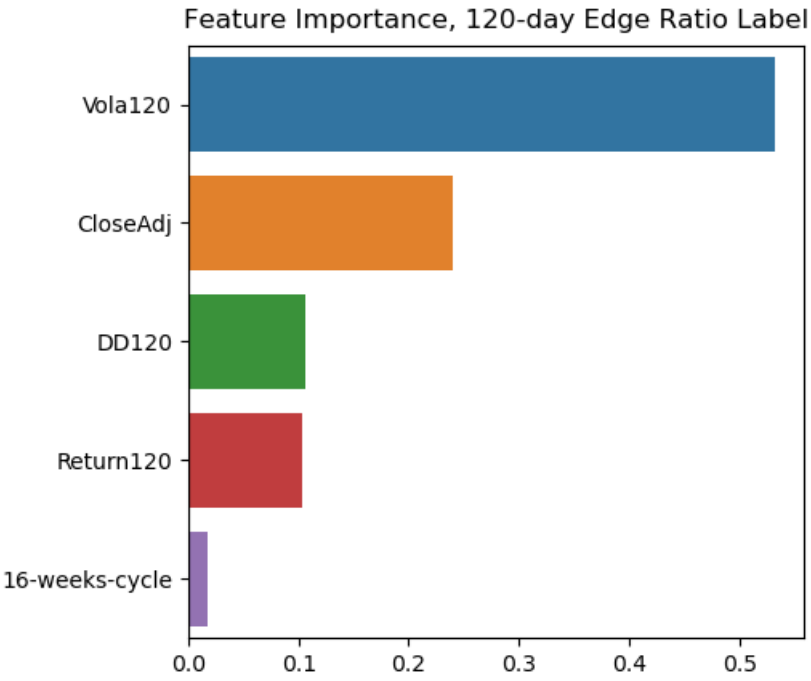

Figure 4.1: The feature importances of a random forest on the S&P 500 data from 1990 to 2017. The 120 indicates that the volatility (Vola), the drawdown (DD) and the return are based on the last 120 trading days.

If the feature importance is low, the feature can be adjusted to better reflect the underlying trading strategy. When this does not help, it is possible that the trading strategy is a false discovery that does not perform well on new data.

To show how to use the feature importance, suppose a new trading strategy should be built. The stock price of Tesla Inc. seems to depend on tweets of its current CEO, Elon Musk. Under the presumption that this is a general rule, a trading strategy is constructed that sells stocks if the corresponding CEO sends negative tweets. To do this, every time a tweet is sent, a sell signal is generated. If a number of characteristics are met, the stock is sold. As described in Section 1.2, machine learning algorithms can be used to analyse the content of written messages. Sentiment analysis can be applied to assign a number that rates the content of a tweet dependent on its sentiment. Given enough additional parameters, like the time a tweet is released, the number of words that are used, how often it is re-tweeted and so on, such a trading strategy can be overfitted easily. Before committing to a trading strategy that does not generalize well on new data, the feature importances of aspects of this trading strategy can be tested. If a feature like the number of words contains no substantial information, it should not be included in the trading strategy, as the danger of overfitting on this feature is potentially high. If the features used in the trading strategy all have a high feature importance and if backtests of this strategy show a high performance, chances are that the trading strategy really works. To summarize in short, rather than trying out an assumption with a backtest directly, an extra step is added to reduce the risk of overfitting. The feature importance can thus work as an additional tool to construct successful trading strategies.

Chapter 5

Building a Trading Strategy

After the dataset has been been prepared in Chapter 2, a model to predict the future stock price movement via labels has been modelled in Chapter 3 and relevant features have been chosen in Chapter 4, it is time to test whether all these machine learning methods can produce a profitable trading strategy. For that purpose, the standard technique to test trading strategies is introduced in Section 5.1. Having established a testing scheme, two major components of a trading strategy will be discussed. How to select profitable assets is the topic of Section 5.2. How to allocate equity to the selected assets in the portfolio will be shown in Section 5.3.

5.1 Backtesting

The goal of every trading strategy is to be profitable in the future. Unfortunately, it is impossible to test the strategy on future data. Even if a strategy would be applied in a small scale today, a profit in two months does not guarantee that there will not be a crash afterwards that the strategy is not prepared for. Therefore, other testing methods, which are called backtests, have to be used to give an impression of how the strategy might perform on future data. It is important to keep in mind that, while a good backtest is desirable, it is worthless if the trading strategy does not also perform well on new data. This is equivalent to giving the numbers of past winning lottery tickets. In the past, the numbers would have been really profitable, but used today, the numbers are not helpful at all. Likewise, it is important not to design an overfitted trading strategy that has a good backtest and performs badly on new data.
The walk-forward backtest uses a historic dataset to simulate how a trading strategy would have performed if it were used in the past. The backtest moves through a dataset from start to finish. At every point in time, the trading strategy only has access to data that have been available at the time. It uses this information to decide what financial instruments to buy or to sell. When the backtest moves ahead a trading day, it simulates how this decision would have played out. Used over a long period of time, this backtesting scheme gives a good impression on how a trading strategy performs on real financial data. This backtesting technique is used in the rest of this thesis.
Apart from walk-forward backtesting, there are other testing schemes that can be used. Some of them use historic datasets, others simulate different scenarios with real or syn-

thetic data. For an extensive discussion of different backtesting techniques, see [López, 2018, pp. 151-192].
When a backtest has been performed, there a multiple statistics to evaluate the performance. For example, keeping track of the start and the end equity is helpful. However, as the start equity can be chosen arbitrarily, it is often better to indicate the relative performance:

$$\text{Total Return} = \frac{S_T}{S_1} - 1,$$

where S_t denotes the equity at $t \in \{1, \dots, T\}$. As the total performance depends on the time period the backtest is performed on, it is a good idea to measure the yearly performance. Suppose a backtest is performed for T trading days. As a year has roughly 252 trading days, the annualized total return can be calculated by:

$$\text{Annualized Return} = (\text{Total Return} + 1)^{\frac{252}{T}} - 1.$$

Furthermore, the volatility of the equity curve can be analysed. Define the daily return of the equity curve for $t \in \{2, \dots, T\}$ as

$$R_t = \frac{S_t}{S_{t-1}} - 1.$$

Then, the annualized volatility can be defined as

$$\text{Annualized Vola} = \sqrt{\frac{1}{T-2} \sum_{t=2}^{T} \left(R_t - \overline{R}\right)^2} \cdot \sqrt{252},$$

where $\overline{R}$ denotes the arithmetic mean. Another useful statistic is the drawdown. The drawdown at $t \in \{1, \dots, T\}$ is defined as

$$\text{DD}_t = \frac{\max\{S_1, \dots, S_t\} - S_t}{\max\{S_1, \dots, S_t\}}.$$

The average drawdown is defined as the arithmetic mean

$$\text{Average DD} = \frac{1}{T} \sum_{t=1}^{T} DD_t$$

and the maximum drawdown is defined as

$$\text{Maximum DD} = \max\left\{\text{DD}_1, \dots, \text{DD}_T\right\}.$$

Finally, a good statistic to combine both risk and reward of a trading strategy is the Sharpe ratio:

$$\text{Sharpe Ratio (Vola)} = \frac{\text{Annualized Return}}{\text{Annualized Vola}}.$$

Instead of using the volatility, the average drawdown can be used as well:

$$\text{Sharpe Ratio (DD)} = \frac{\text{Annualized Return}}{\text{Average DD}}.$$

These statistics will be used to compare backtests in Section 6.3.

5.2 Asset Selection

The core of every trading strategy is the selection of profitable financial instruments. How this selection is carried out differs a lot. For example, fundamental analysis tries to use a business's financial statements to identify profitable stocks. Technical analysis studies past market data to forecast the price development.
The goal of this chapter is to use the machine learning classifier that has been modelled in Chapter 3 in order to select profitable stocks. To compare its performance, it is contrasted to two other selection methods.

5.2.1 Large Market Capitalization

The easiest way to select stocks is to rely on market capitalisation. Following the logic that big companies mostly perform well long-term, companies that make up indices like the S&P 500 are selected exclusively. As the underlying database of this thesis only consists of the assets in the S&P 500 and the STOXX 600 anyway, this selection scheme results in selecting all assets in the dataset. In the following, this serves as a benchmark to compare trading strategies.

5.2.2 Volatility Basket

Another way to select profitable assets is to filter based on the volatility. As it has been shown in Figure 4.1, where different feature importances have been ranked, the volatility seems to be a good indicator for the asset's future price movement. This observation can be used to construct a trading strategy.
High-volatile assets carry a lot of risk, which often results in a loss long-term, see [Van Vliet & De Koning, 2017, pp. 23-27]. That is a reason why two of the three asset allocation methods in Subsection 5.3 try to minimize the portfolio's volatility. Furthermore, it can be shown that the least volatile assets often get outperformed as well. When all assets are sorted based on their volatility from low to high, baskets of assets with similar volatility can be studied. A good sweet spot between risk and reward seems to be the selection of the 2nd volatility quintile, in essence assets with a volatility in between the 20th and the 40th percentile, see [Haase & Platen, 2019, pp. 7-10]. Therefore, the volatility basket strategy used in this thesis always selects all assets in the 2nd volatility quintile. The volatility is based on the daily returns of the last 250 trading days and can be calculated as seen in Subsection 2.2.1.

5.2.3 Machine Learning Selection

The core of the trading strategy explained in this section is the classifier that has been introduced in Chapter 3. Ideally, the classifier has access to a wide range of data, both financial and non-financial. Unfortunately, the scope of this thesis limits the potential of machine learning models, as the features used are almost exclusively based on the close prices of the stocks in the dataset. Nevertheless, a great variety of market indicators can be applied and the model used in this thesis gives a good impression of what a machine learning trading strategy can be capable of. The exact features that are used will be

detailed in Chapter 6.
One possibility to set up a machine learning trading strategy is to train a classifier on a dataset and to use its predictions to identify assets to invest in. For that purpose, the trained classifier is given a set of current, unlabelled samples of financial instruments. If the trading strategy was to go live, new samples would have to be created from financial data by calculating the features, which only need access to past data. Labelling these current samples is impossible, as this would require future knowledge. However, labelled samples are only necessary in order to train a model, not to give predictions. For example, in order to train a decision tree, a labelled dataset is necessary. Once the tree is build, it can be used to give predictions for new, unlabelled samples.
When a backtest is performed, creating a current sample is simulated by stripping a sample in the test dataset off its label. Based on the predicted label of the current sample, the corresponding asset is added to the current portfolio. When assets have been labelled -1 and 1, all assets with samples that are predicted to have a positive label are selected for the portfolio.
Alternatively, the predicted probability that an asset belongs to the positive class can be used to select assets to invest in. As explained in Subsection 3.4.1, the output of classifiers contains a value $p \in [0, 1]$. If this value is larger than $\frac{1}{2}$, a positive label is predicted. By selecting a minimal threshold $p_{\min} \in [0, 1]$, it is possible to only invest in assets that are very likely to have a good price development. For that purpose, a high threshold has to be selected. Thus, the machine learning model foregoes assets with a predicted ambiguous price development in favour of a few strong assets. The exact threshold $p_{\min}$ greatly depends on the underlying classifier, as reported values for p can vary a lot.
As explained in Section 3.4, an ensemble of different models often improves the overall performance. Therefore, instead of relying on one classifier, the trading strategy in this thesis uses multiple classifiers that are fitted on the same training dataset. To predict the label of an unlabelled sample, the predictions of all classifiers are aggregated via the soft voting method. If the prediction is positive, the corresponding asset is selected and becomes a part of the portfolio. Alternatively, a minimal threshold $p_{\min}$ has to be surpassed in order to qualify for the portfolio.
Theoretically, it is possible to use any classifier in the ensemble method explained above. In fact, the more diverse classifiers are added, the better the ensemble is expected to perform. That means that there does not have to be a decision between random forests, neural networks or other classifiers, as all models can be used simultaneously. In the work for this thesis, random forests have proven to be a reliable classifier for financial data. Therefore, multiple random forests are used in this thesis. To diversify the individual random forests, different hyperparameters are chosen. Furthermore, the random forests are fitted on data that have been labelled differently, for example by selecting varying labelling horizons. Good combinations of hyperparameters are determined via the cross-validation method on the training set, which will be discussed in Section 6.2.

5.3 Asset Allocation

When the assets to invest in have been chosen, the natural question is how to allocate the equity to the assets. Investing in a single stock promises a high reward if the stock price of that particular asset rises. However, a falling stock price results in heavy losses. Given how hard it is to accurately predict stock price movements, such a strategy is close to gambling. A better idea is to spread investments over multiple assets. In the beginning of modern portfolio theory, Harry Markowitz showed the merit of diversification mathematically. He even went as far as to say that "a rule of behaviour which does not imply the superiority of diversification must be rejected both as a hypothesis and a maxim", because "diversification is both observed and sensible", see [Markowitz, 1952, p. 77].
There are several ways to diversify a portfolio, which will be discussed in the following. As seen in many other areas, machine learning techniques offer new approaches for classic problems.

5.3.1 Equal Weight

The most intuitive way to allocate the equity is to spread it on all assets evenly. After a certain time, high performing assets allocate lots of equity, while the proportion of low performing stocks falls. It would be tempting to keep this structure indefinitely. However, over time, this process of accumulation makes the portfolio prone to a sudden drop in stock prices of a few assets, which is the reason a diversification was carried out in the first place. Hence, after a certain period of time, the portfolio should be reallocated. Solely relying on the equal weight allocation method can be problematic, because of the correlation of assets in the portfolio. For example, in a portfolio of ten different assets, there might be three stocks of international banks. On paper, the portfolio is invested in ten different assets, but almost a third of the equity is invested in banks. If a financial crisis occurs, all three assets are likely to drop simultaneously, resulting in heavy losses of the portfolio. That means that the portfolio has not been diversified properly.

5.3.2 Mean-Variance

To address this issue, techniques of classical areas of mathematics, like algebra or calculus, have been used to exploit the correlation structure of the portfolio in order to derive a diversified portfolio. The idea is to optimize the weights in such a way that the variance of the portfolio is minimized, while generating a predefined expected return.
Let $(S_{t,j})_{t\in\{1,\dots,T\}}$ denote the price series for an asset $j \in \{1,\dots,N\}$. The daily return can be calculated by

$$R_{t,j} = \frac{S_{t,j} - S_{t-1,j}}{S_{t-1,j}}.$$

Furthermore, suppose the time series $R_{\cdot,j} = \left(R_{t,j}\right)_{t\in\{2,\dots,T\}}$ is stationary. In essence, that means that the returns can be interpreted as observations of an underlying, unchanging

random variable

$$\boldsymbol{R}_j : \Omega \to \mathbb{R}, \quad \omega_t \mapsto \boldsymbol{R}_j(\omega_t) = R_{t,j}$$

for an appropriate probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and $j \in \{1, \dots, N\}$. The returns of all assets can thus be interpreted as a random vector

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R}_1 \\ \vdots \\ \boldsymbol{R}_N \end{bmatrix}.$$

With $w \in \mathbb{R}^N$ denoting the portfolio weights and $\mu \in \mathbb{R}$ as a threshold for the expected return, the trade-off between the expected return and the variance can be defined as

$$\begin{aligned} \min_{w \in \mathbb{R}^N} \quad & \operatorname{Var}\left[w^T \boldsymbol{R}\right] \\ \text{s.t.} \quad & \sum_{j=1}^{N} w_j = 1 \\ & w_j \geq 0, \quad j \in \{1, \dots, N\} \\ & \mathrm{E}\left[w^T \boldsymbol{R}\right] \geq \mu \end{aligned} \tag{5.1}$$

There is no formula for an explicit solution to this problem. However, if the constraint of positive weights is loosened, there are formulas for the optimal portfolio, for example see [Maier-Paape & Zhu, 2018, pp. 19-21]. When it is impossible to use an explicit formula, optimisation algorithms have to be utilized.
In theory, by minimizing the portfolio variance directly, it is possible to prevent big losses, thus generating a diversified portfolio. However, the optimized portfolios suffer from a lot of problems empirically, see [López, 2018, pp. 221-223]. Firstly, the portfolio weight is usually concentrated in a few assets. In general, this leads to a portfolio that is prone to a price drop of these key assets, which means the portfolio is not diversified properly. To counter this, it is a good idea to set constraints for maximal weights. Secondly, portfolios tend not to be stable, a small change in the return series can result in a completely different portfolio structure. Thirdly, standard mean-variance portfolios usually underperform and can be beaten by the simple equal weight strategy described in Subsection 5.3.1, see [De Miguel, 2009, p. 1931].
A reason why this method fails to reproduce the theoretical results in practice could be computational. Rearranging the equation shows that the covariance matrix

$$\Sigma = \mathrm{E}\left[\left(\boldsymbol{R} - \mathrm{E}\left[\boldsymbol{R}\right]\right)\left(\boldsymbol{R} - \mathrm{E}\left[\boldsymbol{R}\right]\right)^T\right] \in \mathbb{R}^{N \times N}$$

has to be estimated, because

$$\operatorname{Var}\left[w^T \boldsymbol{R}\right] = w^T \Sigma w \in \mathbb{R}.$$

With $R_{t,\cdot} \in \mathbb{R}^{1 \times N}$ for $t \in \{1, \dots, T\}$ and $\overline{R} = \frac{1}{T} \sum_{t=1}^{T} R_{t,\cdot} \in \mathbb{R}^{1 \times N}$ as the sample mean, the covariance matrix can be estimated by

$$V = \frac{1}{T-1} \sum_{t=1}^{T} \left(R_{t,\cdot} - \overline{R}\right)^T \left(R_{t,\cdot} - \overline{R}\right) \in \mathbb{R}^{N \times N}. \tag{5.2}$$

In general, quadratic programming methods that try to solve Equation (5.1) require the inversion of the covariance matrix, for example see [Markowitz, 1956, pp. 111-133] or [Nocedal & Wright, 2006, pp. 463-490]. An optimisation with a covariance matrix that is not invertible is challenging, as there can be several stationary points and local minima, see [Nocedal & Wright, 2006, p. 449]. However, to estimate an invertible covariance matrix, there have to be a lot of samples. In general, at least $\frac{1}{2}N(N+1)$ independent and identically distributed (iid) observations are necessary for an estimation that is not singular, see [López, 2018, p. 223]. For a portfolio of 50 assets, that means that there have to be at least five years of daily iid data. Furthermore, the correlation structure does not remain invariant with a reasonable confidence level for such periods of time.
To understand why it is problematic to invert the covariance matrix, it is helpful to look at its condition number. The condition number shows how sensitive a function is to small changes or errors in the input, see [Dahmen & Reusken, 2006, p. 11]. The condition number is lowest for a diagonal correlation matrix, which is its own inverse. The more correlated assets are added, the higher the condition number rises and numerical errors make the inverse unstable. If there are many correlated assets, small changes in the underlying return matrix produce a completely different inverse. At some point, the covariance matrix is not invertible anymore. This means that the more correlated the assets in the portfolio are, the higher numerical errors occur, which leads to a bad diversification when it is needed the most.

5.3.3 Hierarchical Risk Parity

To solve this issue, new methods from graph theory and machine learning can be used. Until now, most of macroeconomic and financial research rests upon classical applications of linear algebra and stochastic calculus. Linear algebra focuses on the location in Euclidean geometry, stochastic calculus on the changes of this location in conjunction with an underlying random variable. An examination of the locations of the prices in a complex system like the financial market is very important. However, logical relations can help to describe this structure as well. The financial market could be understood as a network of interconnected assets that influence each others price movements. Linear algebra can answer questions about how closely together the system moves, but not what connections ensure the flow in the network or what nodes can shut the network off. A good example is a subway map. In it, the exact coordinates of stations are discarded to focus on the relations of train stations. To answer questions about a system's organization, areas of mathematics like graph theory are helpful. In combination with machine learning, new approaches to investigate financial markets can be investigated, see [Calkin & López, 2014, p. 43-45]. These methods can complement existing asset allocation methods.
The approach introduced in this thesis is Hierarchical Risk Parity (HRP). Using a clustering algorithm, hierarchical structures among the assets are exploited to circumvent the stability issues described earlier.

Hierarchical Clustering

A covariance matrix includes the covariances of every pair of return series in the dataset. This translates to a fully connected graph, where every node is connected to all other nodes, see Figure 5.1. In the financial context, that means that every asset is a full substitute for any other asset. In reality, an investor is more likely to exchange a European utility company with another European utility company, rather than exchanging it with an Asian fintech start-up. Moreover, inverting this matrix means evaluating every single connection, which is very complex.

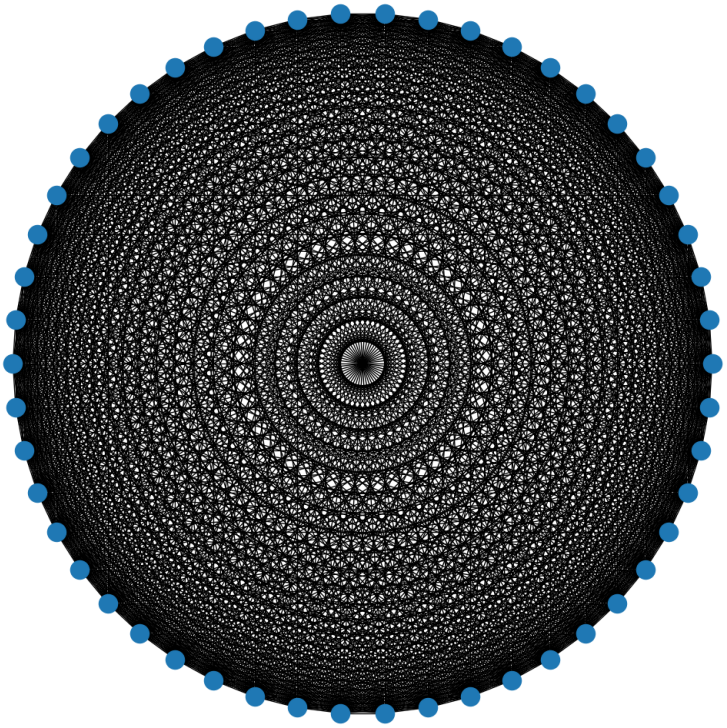

Figure 5.1: A portfolio of 50 assets displayed as a fully connected graph.

In graph theory, there is another graph structure that is applicable for this scenario: a tree graph, as seen in Figure 5.2. In it, nodes that are similar are connected via an edge. When two nodes have been connected, they form a cluster. When another node is close to a node in the cluster, an edge between those nodes is added and the cluster thus expanded. By keeping track of the order nodes are added to clusters, a tree graph naturally gives its nodes a hierarchy. In the financial context, German fintechs could be grouped together, which could in turn be part of the cluster of European financial services and so on.
In contrast to a fully connected graph, the tree graph contains information about the nodes hierarchy. Nevertheless, the structure is simpler and there are fewer connections that have to be evaluated when inverting covariances. While a fully connected graph has $\frac{1}{2}N(N-1)$ edges, the proposed tree structure only has $N-1$.
To derive this structure, the hierarchical clustering algorithm can be used, see [López,

2018, pp. 224-229]. The algorithm starts by treating every asset as a single cluster containing itself. Then, it merges the closest assets into a bigger cluster one by one, until all assets are grouped together. In order to do this, the criteria that make up close assets have to be defined.

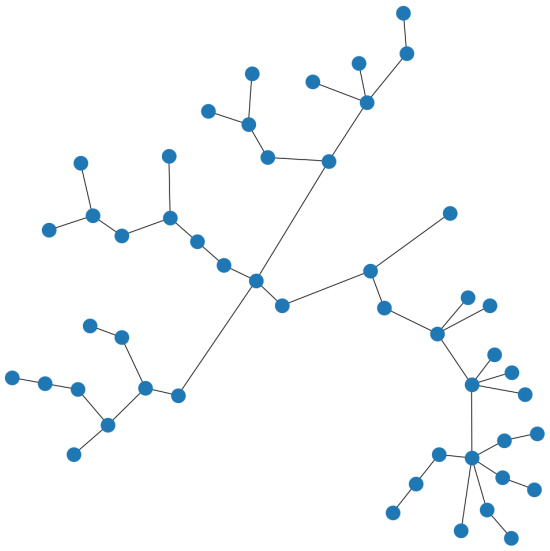

Figure 5.2: A portfolio of 50 assets displayed as a tree graph. When to connect two assets will be described later in detail. For example, an edge can indicate that the return series of the two linked assets correlate the most out of all assets.

In finance, it makes sense to group assets if their return series are highly correlated. Let $R \in \mathbb{R}^{T\times N}$ be a return matrix. As described in Subsection 5.3.2, the columns can be interpreted as the realisations of underlying random variables

$$\boldsymbol{R}_j : \Omega \to \mathbb{R}, \quad \omega_t \mapsto \boldsymbol{R}_j(\omega_t) = R_{t,j}$$

for $j \in \{1,\dots,N\}$ and $t \in \{1,\dots,T\}$. Furthermore, let V be the estimated covariance matrix as described in (5.2). For $j \in \{1,\dots,N\}$, the empirical standard deviation of $\boldsymbol{R}_j$ can be described by

$$\sigma_{\boldsymbol{R}_j} = \sqrt{V_{j,j}}$$

For $i,j \in \{1,\dots,N\}$, let

$$\rho_{i,j} = \frac{V_{i,j}}{\sigma_{\boldsymbol{R}_i}\sigma_{\boldsymbol{R}_j}} \in [-1,1] \tag{5.3}$$

denote Pearson's correlation coefficient of $\boldsymbol{R}_i$ and $\boldsymbol{R}_j$. Then, the correlation matrix can be expressed as

$$\rho = \left[\rho_{i,j}\right]_{i,j\in\{1,\dots,N\}}.$$

Based on this correlation, a distance measure can be defined:

$$\widetilde{d} : \{1,\dots,N\}^2 \to [0,1] \subset \mathbb{R}, \quad (i,j) \mapsto \widetilde{d}_{i,j} = \sqrt{\frac{1}{2}(1-\rho_{i,j})}.$$

This distance measure is a metric, see [López, 2018, p. 239] for a proof. It is low if two return series are correlated and high if they are anticorrelated. To give an overview, all distances of assets are combined in a single distance matrix

$$\widetilde{D} = \left[\widetilde{d}_{i,j}\right]_{i,j\in\{1,\dots,N\}}.$$

When two assets are clustered together, distances to other assets should be considered as well. This prevents a huge central cluster surrounded by outliers and enables the formation of many equally sized clusters. In order to do that, the distance matrix is updated by calculating the Euclidean distance between its columns. For two columns $\widetilde{D}_i$ and $\widetilde{D}_j$ of $\widetilde{D}$, the Euclidean distance is defined as

$$\begin{aligned}\mathring{d} : \mathbb{R}^N \times \mathbb{R}^N &\to \left[0, \sqrt{N}\right] \subset \mathbb{R},\\ \left(\widetilde{D}_i, \widetilde{D}_j\right) &\mapsto \mathring{d}_{i,j} = \left|\left|\widetilde{D}_i - \widetilde{D}_j\right|\right|_{\mathbb{R}^N} = \sqrt{\sum_{n=1}^{N}\left(\widetilde{d}_{n,i} - \widetilde{d}_{n,j}\right)^2}.\end{aligned}$$

The new distance matrix, which is a distance matrix of distances, can be defined as

$$\mathring{D} = \left[\mathring{d}_{i,j}\right]_{i,j\in\{1,\dots,N\}}.$$

This matrix serves as a base distance matrix for all following calculations.
The hierarchical clustering algorithm forms clusters by grouping several assets together. At the start, every asset is part of a one-element cluster. The set of clusters at the beginning can be defined as

$$U^{(1)} = \{u_1, \dots, u_N\},$$

with $u_i = \{i\}$ for $i \in \{1, \dots, N\}$. New clusters are created by merging clusters that are close. If the clusters u_{i^*} and u_{j^*} are merged in the kth iteration of the algorithm, the set of clusters is updated:

$$U^{(k+1)} = \left(U^{(k)} \backslash \{u_{i^*}, u_{j^*}\}\right) \cup \{u_{N+k}\},$$

with $u_{N+k} = u_{i^*} \cup u_{j^*} \subset \{1, \dots, N\}$. In order to identify clusters to merge, a distance measure of clusters has to be defined, which is called 'linkage criterion'. Here, the single linkage criterion is used, which is defined as

$$d : \mathcal{P}\left(\{1, \dots, N\}^2\right) \to \mathbb{R}, \quad \left(u_i, u_j\right) \mapsto d(u_i, u_j) = \min\left\{\mathring{d}_{\alpha,\beta} : \ \alpha \in u_i, \beta \in u_j\right\},$$

where $\mathcal{P}$ denotes the power set. The single linkage criterion could be applied on a distance matrix of assets like $\widetilde{D}$ with an alternative definition

$$d'(u_i, u_j) = \min\left\{\widetilde{d}_{\alpha,\beta} : \ \alpha \in u_i, \beta \in u_j\right\}.$$

In that case, the linkage criterion is very intuitive, as the distance is simply the minimum distance between assets of each cluster. However, this might produce stretched out clusters. To counter this flaw, it is applied on a distance matrix of distances $\mathring{D}$, which

takes into account multiple distances $\widetilde{d}$ between assets. There are many other linkage criteria to choose from, each producing different clusters, see [Raffinot, 2018, p. 91].
To give an overview of the distances in between clusters, all distances in the kth iteration of the algorithm are collected in a set

$$D^{(k)} = \left\{ d(u_i, u_j) : \ u_i, u_j \in U^{(k)} \right\}.$$

As the number of clusters in $U^{(k)}$ is finite, this set can be displayed as a matrix. In fact, because all clusters only contain one element in the beginning, $D^{(1)}$ can be mapped to $\mathring{D}$. In the following, the set $D^{(k)}$ is called the distance matrix of clusters in the kth iteration. In every iteration k, the minimal entry of the corresponding distance matrix $D^{(k)}$ is used to determine the clusters that are merged:

$$(u_{i^*}, u_{j^*}) = \underset{\substack{(u_i,u_j)\in U^{(k)}\times U^{(k)} \\ u_i \neq u_j}}{\operatorname{argmin}} \left\{ d(u_i, u_j) \right\} \quad \Rightarrow \quad u_{N+k} = u_{i^*} \cup u_{j^*}.$$

Of course, it is possible that there are multiple clusters with identical distances. In fact, as the distance matrix is symmetric, there are always at least two identical minimal entries. In these cases, the pair of clusters to merge can be chosen randomly out of the clusters with minimal distance. The order is not that relevant, because the other pair of clusters will be merged in the next iteration anyway, at least if the single linkage criterion is used.
When two clusters have been merged, the algorithm updates the list of clusters $U^{(k+1)}$ with u_{N+k}, calculates new entries for the distance matrix $D^{(k+1)}$ and merges two more clusters. This is repeated until all assets form a single big cluster.
The full hierarchical clustering algorithm can be summarized as follows:

1. Initialize the list of clusters:

   $$U^{(1)} = \{u_1, \ldots, u_N\},$$

   with $u_i = \{i\}$ for $i \in \{1, \ldots, N\}$.

2. Set the iteration counter:

   $$k = 1.$$

3. Transform the correlation matrix to a distance matrix of assets with Pearson's correlation coefficient $\rho_{i,j}$ as defined in (5.3):

   $$\rho \to \widetilde{D} \quad \text{with} \quad \widetilde{d}_{i,j} = \sqrt{\frac{1}{2}(1 - \rho_{i,j})}.$$

4. Transform this distance matrix by computing the Euclidean distance in between columns:

   $$\widetilde{D} \to \mathring{D} \quad \text{with} \quad \mathring{d}_{i,j} = \sqrt{\sum_{n=1}^{N} \left(\widetilde{d}_{n,i} - \widetilde{d}_{n,j} \right)^2}.$$

5. If $\left|U^{(k)}\right| = 1$, stop.

6. Calculate the distance matrix of clusters

$$\mathring{D} \to D^{(k)} = \left\{ d(u_i, u_j) : \; u_i, u_j \in U^{(k)} \right\}$$

with the linkage criterion

$$d(u_i, u_j) = \min \left\{ \mathring{d}_{\alpha,\beta} : \; \alpha \in u_i, \beta \in u_j \right\}.$$

7. Identify the two clusters (u_{i^*}, u_{j^*}) that are closest and denote them as cluster u_{N+k}:

$$(u_{i^*}, u_{j^*}) = \underset{\substack{(u_i,u_j)\in U^{(k)}\times U^{(k)} \\ u_i \neq u_j}}{\operatorname{argmin}} \left\{ d(u_i, u_j) \right\} \quad \Rightarrow \quad u_{N+k} = u_{i^*} \cup u_{j^*}.$$

8. Update the list of clusters:

$$U^{(k+1)} = \left(U^{(k)} \backslash \{u_{i^*}, u_{j^*}\} \right) \cup \{u_{N+k}\}.$$

9. Increase the iteration counter k by 1 and loop to step 5.

In the end, the history of the list of clusters can be used to derive a tree graph, which can be visualized as seen in the example in Figure 5.3.

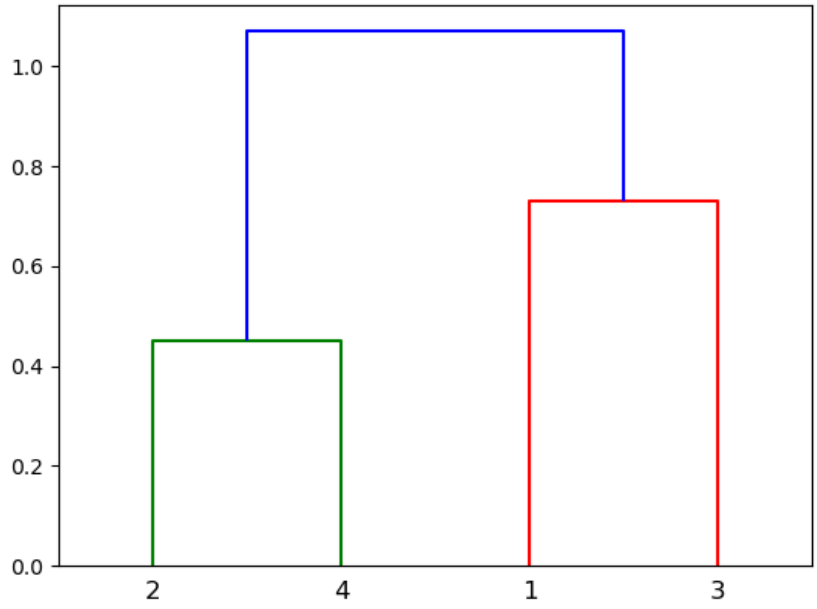

Figure 5.3: A dendogram of the example that is about to be discussed. On the y-axis, the distance between merging clusters is displayed.

Hierarchical Clustering - Example

To properly understand the hierarchical clustering algorithm, it is helpful to test it in an example. For that purpose, suppose that there are four assets with a corresponding correlation matrix

$$\rho = \begin{bmatrix} 1 & -0.3 & 0.5 & -0.4 \\ -0.3 & 1 & 0.1 & 0.8 \\ 0.5 & 0.1 & 1 & 0 \\ -0.4 & 0.8 & 0 & 1 \end{bmatrix}$$

Step 1: Initialize the list of clusters:

$$U^{(1)} = \{u_1, u_2, u_3, u_4\} = \{\{1\}, \{2\}, \{3\}, \{4\}\}.$$

Step 2: Set the iteration counter $k = 1$.

Step 3: Derive a distance matrix of assets from the correlation matrix:

$$\rho = \begin{bmatrix} 1 & -0.3 & 0.5 & -0.4 \\ -0.3 & 1 & 0.1 & 0.8 \\ 0.5 & 0.1 & 1 & 0 \\ -0.4 & 0.8 & 0 & 1 \end{bmatrix} \rightarrow \widetilde{D} = \begin{bmatrix} 0 & 0.8062 & 0.5 & 0.8367 \\ 0.8062 & 0 & 0.6708 & 0.3162 \\ 0.5 & 0.6708 & 0 & 0.7071 \\ 0.8367 & 0.3162 & 0.7071 & 0 \end{bmatrix}$$

Step 4: Transform the distance matrix via the Euclidean distance in between columns:

$$\widetilde{D} = \begin{bmatrix} 0 & 0.8062 & 0.5 & 0.8367 \\ 0.8062 & 0 & 0.6708 & 0.3162 \\ 0.5 & 0.6708 & 0 & 0.7071 \\ 0.8367 & 0.3162 & 0.7071 & 0 \end{bmatrix}$$

$$\rightarrow \mathring{D} = \begin{bmatrix} 0 & 1.2649 & 0.7315 & 1.2973 \\ 1.2649 & 0 & 1.0708 & 0.4497 \\ 0.7315 & 1.0708 & 0 & 1.1131 \\ 1.2973 & 0.4497 & 1.1131 & 0 \end{bmatrix}$$

Step 5: $\left|U^{(1)}\right| > 1$, so continue.

Step 6: Use the linkage criterion to derive the distance matrix of clusters. In the first iteration, $D^{(1)} = \mathring{D}$.

Step 7: Use the smallest non-diagonal entry of the distance matrix $D^{(1)}$ to identify clusters to merge:

$$0.4497 = d(\{2\}, \{4\}) \quad \Rightarrow \quad u_5 = \{2, 4\}.$$

Step 8: Update the list of clusters:

$$U^{(1)} = \{\{1\}, \{2\}, \{3\}, \{4\}\} \quad \rightarrow \quad U^{(2)} = \{\{1\}, \{3\}, \{2, 4\}\}.$$

Step 9: Update $k = 2$ and loop to step 5.

Step 5′: $\left|U^{(2)}\right| > 1$, so continue.

Step 6′: Update the distance matrix of clusters. Start by calculating the linkage criterion $d(u_i, u_j) = \min\left\{\mathring{d}_{\alpha,\beta} : \alpha \in u_i, \beta \in u_j\right\}$ with $u_i, u_j \in U^{(2)}$ for the new cluster:

$$\begin{bmatrix} d(\{1\},\{2,4\}) \\ d(\{3\},\{2,4\}) \end{bmatrix} = \begin{bmatrix} 1.2649 \\ 1.0708 \end{bmatrix}.$$

The list of clusters $U^{(2)}$ has been updated by removing the clusters $\{2\}$ and $\{4\}$ and by adding the merged cluster $\{2,4\}$. Thus, to calculate the new distance matrix of clusters $D^{(2)}$, the rows and columns that correspond to $\{2\}$ and $\{4\}$ are deleted and a new row and column for $\{2,4\}$ is added:

$$D^{(1)} = \begin{bmatrix} 0 & 1.2649 & 0.7315 & 1.2973 \\ 1.2649 & 0 & 1.0708 & 0.4497 \\ 0.7315 & 1.0708 & 0 & 1.1131 \\ 1.2973 & 0.4497 & 1.1131 & 0 \end{bmatrix}$$

$$\rightarrow D^{(2)} = \left[\begin{array}{cc:c} 0 & 0.7315 & 1.2649 \\ 0.7315 & 0 & 1.0708 \\ \hdashline 1.2649 & 1.0708 & 0 \end{array}\right]$$

Step 7′: Use the smallest non-diagonal entry of the distance matrix $D^{(2)}$ to identify clusters to merge:

$$0.7315 = d(\{1\},\{3\}) \quad \Rightarrow \quad u_6 = \{1,3\}.$$

Step 8′: Update the list of clusters:

$$U^{(2)} = \{\{1\},\{3\},\{2,4\}\} \quad \rightarrow \quad U^{(3)} = \{\{2,4\},\{1,3\}\}.$$

Step 9′: Update $k = 3$ and loop to step 5.

Step 5″: $\left|U^{(3)}\right| > 1$, so continue.

Step 6″: Update the distance matrix of clusters. Start by calculating the linkage criterion for the new cluster:

$$d(\{2,4\},\{1,3\}) = 1.0708$$

Delete the rows and columns of $D^{(2)}$ that correspond to $\{1\}$ and $\{3\}$ and add a new row and column for $\{1,3\}$:

$$D^{(2)} = \begin{bmatrix} 0 & 0.7315 & 1.2649 \\ 0.7315 & 0 & 1.0708 \\ 1.2649 & 1.0708 & 0 \end{bmatrix} \rightarrow D^{(3)} = \left[\begin{array}{c:c} 0 & 1.0708 \\ \hdashline 1.0708 & 0 \end{array}\right]$$

Step 7″: Use the smallest non-diagonal entry of the distance matrix $D^{(3)}$ to identify clusters to merge:

$$1.0708 = d(\{2,4\},\{1,3\}) \quad \Rightarrow \quad u_7 = \{2,4,1,3\}.$$

Step 8″: Update the list of clusters:

$$U^{(3)} = \{\{2,4\},\{1,3\}\} \quad \rightarrow \quad U^{(4)} = \{\{2,4,1,3\}\}.$$

Step 9″: Update $k = 4$ and loop to step 5.

Step 5‴: $\left|U^{(4)}\right| = 1$, so stop.

The result of the hierarchical clustering algorithm can be visualized with a dendogram, see Figure 5.3. A dendogram plots the order in which clusters are formed, which assets a cluster consists of and how close they are. In this example, it is apparent that the assets 2 and 4 and, with some limitations, the assets 1 and 3 can be grouped together.

Quasi-Diagonalization

By using the results of the hierarchical clustering algorithm, the assets can be reordered. The new order, which can be seen on the x-axis in Figure 5.3, can be derived by following the tree graph in the dendogram from top to bottom:

$$\begin{bmatrix} u_7 \end{bmatrix} \rightarrow \begin{bmatrix} u_5 \\ u_6 \end{bmatrix} \rightarrow \begin{bmatrix} u_5 \\ 1 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} 2 \\ 4 \\ 1 \\ 3 \end{bmatrix}.$$

If the return series of two assets correlate, they are placed together. Thus, when the correlation matrix is rearranged accordingly, it is visible that the highest correlation values are located near the diagonal of the matrix, see Figures 5.4 and 5.5. Clusters are displayed clearly as rectangles.
This order is then used to rearrange the covariance matrix, which will be used when assigning portfolio weights.

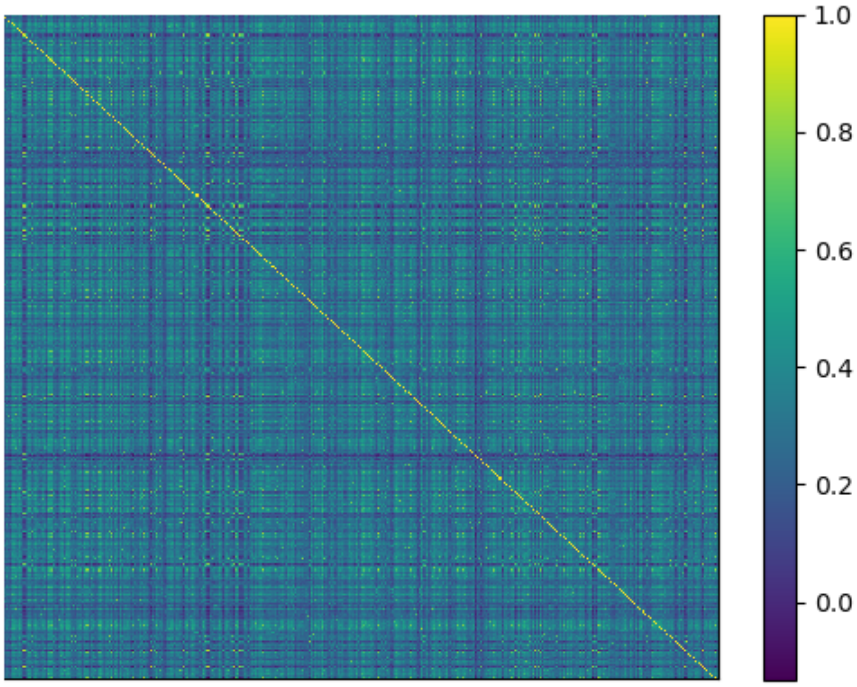

Figure 5.4: The correlation matrix of the S&P 500 companies before clustering. Underlying is a dataset of daily returns from 2014 to 2019.

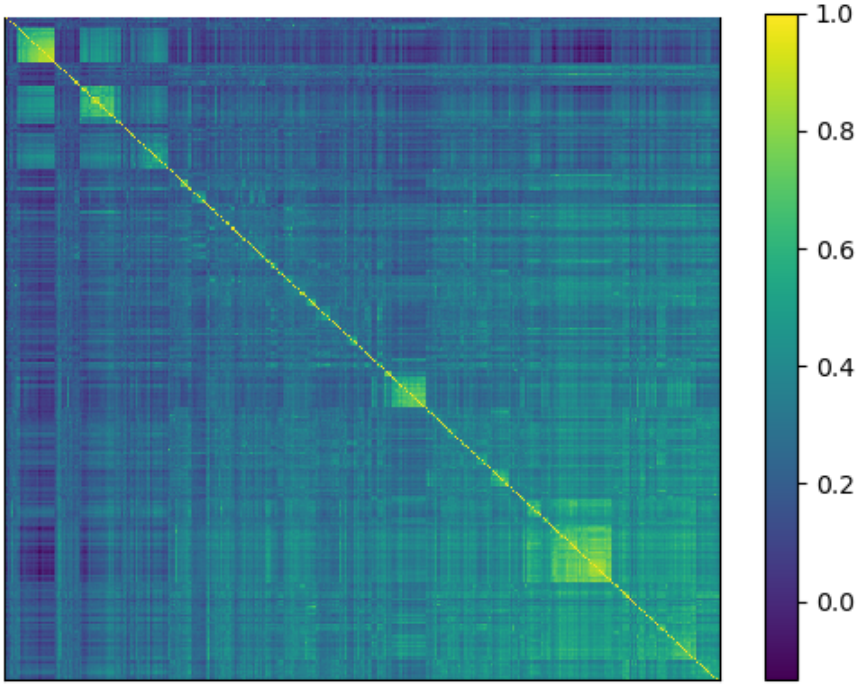

Figure 5.5: The correlation matrix of the S&P 500 companies after clustering. Underlying is a dataset of daily returns from 2014 to 2019.

Recursive Bisection

Clustering has delivered a quasi-diagonal matrix. For a diagonal covariance matrix, the inverse-variance allocation is optimal. To see this, consider the standard portfolio optimization problem with the estimated covariance matrix V:

$$\begin{aligned} \min_{w \in \mathbb{R}^N} \quad & w^T V w \\ \text{s.t.} \quad & w^T 1_N = 1, \end{aligned}$$

where

$$1_N = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^N.$$

Using a Lagrange multiplier, it can be easily shown that the solution to this problem is

$$w = \frac{V^{-1} 1_N}{1_N^T V^{-1} 1_N} \in \mathbb{R}^N. \tag{5.4}$$

A diagonal covariance matrix can be inverted by inverting the values on its diagonal. Therefore, if the covariance matrix is diagonal, the optimal weight for $j \in \{1, \ldots, N\}$ can be calculated by

$$w_j = \frac{V_{j,j}^{-1}}{\sum_{n=1}^{N} V_{n,n}^{-1}}.$$

For $N = 2$, the formula reads

$$w_1 = \frac{\frac{1}{V_{1,1}}}{\frac{1}{V_{1,1}} + \frac{1}{V_{2,2}}} = 1 - \frac{V_{1,1}}{V_{1,1} + V_{2,2}} \quad \text{and} \quad w_2 = \frac{V_{1,1}}{V_{1,1} + V_{2,2}}.$$

The following algorithm uses a recursive bisection to set the weights. This means that at every recursive step, the task is to allocate the weights to two subsets of assets at a time. The allocation method used to do this is inspired by the optimal allocation formula for $N = 2$.

To explain how the weight is distributed, the first step is explained in detail, using a few assumptions to describe an ideal weight allocation. These constraints will be relaxed later.

Let $V \in \mathbb{R}^{N \times N}$ denote the covariance matrix that has been rearranged with the hierarchical clustering algorithm. Suppose that the set of assets is split in half. That means that

$$L_1 = \left\{1, \ldots, \left\lfloor \frac{N}{2} \right\rfloor \right\} \text{ and } L_2 = \left\{ \left\lfloor \frac{N}{2} \right\rfloor + 1, \ldots, N \right\},$$

where $\lfloor \cdot \rfloor$ denotes the floor function. The task is to allocate weights to L_1 and L_2. In order to simplify the first step, the assets in their respective subsets L_1 and L_2 are treated as a unity. Basically, the assumption is that L_1 and L_2 form homogeneous clusters. In that case, let $v_L \in \mathbb{R}$ denote the 'variance' of a cluster L, which will be defined later. For example, the mean variance of a cluster could be used as v_L. Furthermore, suppose

that the assets in their respective cluster are completely independent from the assets in the other cluster, basically $V_{i,j} = 0$ for all assets $i \in L_1$ and $j \in L_2$. Under these assumptions, the task to reallocate the weight between the assets can be simplified to the task of allocating the weight to the two clusters L_1 and L_2 that, if treated as a unity, have a covariance matrix

$$\begin{bmatrix} v_{L_1} & 0 \\ 0 & v_{L_2} \end{bmatrix} \in \mathbb{R}^{2\times 2}.$$

Thus, following the formula for optimal weights for $N = 2$, the weights for the assets can be set as

$$\begin{aligned} w_i &= 1 - \frac{v_{L_1}}{v_{L_1} + v_{L_2}} \in [0,1] \quad \text{for } i \in L_1, \\ w_j &= \frac{v_{L_1}}{v_{L_1} + v_{L_2}} \in [0,1] \quad \text{for } j \in L_2. \end{aligned}$$

It should be noted that the weights do not add up to 1 anymore, because every asset is given the weight of the whole cluster. However, because of the recursive bisection, the weights will be altered in the next step. In the end, the constraint $w^T 1_N = 1$ is met again.

Of course, the assumptions of homogeneous and independent clusters are not realistic. However, the rearrangement by the hierarchical clustering algorithm has created a quasi-diagonal covariance matrix, where the highest values are located in the proximity of the diagonal entries. That means that, although the covariance $V_{i,j}$ of assets $i \in L_1$ and $j \in L_2$ is not zero, it is generally small. Furthermore, assets in L_1 and L_2 are most likely not homogeneous, but assets that have been placed together in clusters might correlate strongly nonetheless. As the assumptions are not met in general, this allocation method does not grand optimal weights, but it might be a good heuristic allocation. Other methods to allocate weights to two subsets of assets are applicable.

To describe the variance of a subset of assets v_L, let $V \in \mathbb{R}^{N\times N}$ denote the covariance matrix that has been rearranged with the hierarchical clustering algorithm. For a contiguous subset $L \subset \{1, \dots, N\}$ with $|L| = n$, the covariance matrix between the constituents of L can be defined as

$$V_L = \left[V_{i,j}\right]_{i,j\in L} \in \mathbb{R}^{n\times n}.$$

Of course, the order of assets is preserved.

There are multiple ways to define the variance of a subset of assets. For example, the mean variance of all assets could be chosen:

$$\overline{v}_L = \frac{1}{n} 1_n^T \operatorname{diag}\left[V_L\right] 1_n,$$

where

$$\operatorname{diag}[A] = \begin{bmatrix} A_{1,1} & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & A_{n,n} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

denotes the diagonal matrix for a matrix $A \in \mathbb{R}^{n \times n}$.
The algorithm in this thesis uses the variance of an inverse-variance allocation as described in Equation (5.4), applied to V_L:

$$v_L = \widetilde{w}_L^t V_L \widetilde{w}_L,$$

with

$$\widetilde{w}_L = \frac{\operatorname{diag}[V_L]^{-1} 1_n}{1_n^T \operatorname{diag}[V_L]^{-1} 1_n}.$$

In the calculation, assets with low variance receive more weight. Moreover, if two assets i and j in L both have a low variance, the covariance $V_{i,j}$ is given much weight as well.
The complete algorithm can be summarized as follows:

1. The algorithm is initialized by:
   (a) Setting the list of items $L = \{L_0\}$, with $L_0 = \{1, \ldots, N\}$.
   (b) Assigning a unit weight to all items: $w_j = 1$ for all $j \in \{1, \ldots, N\}$.
2. If $|L_i| = 1$ for all $L_i \in L$, stop.
3. For each $L_i \in L$ such that $|L_i| > 1$:
   (a) Bisect L_i into two contiguous subsets $L_i^1 \cup L_i^2 = L_i$, where $\left|L_i^1\right| = \left\lfloor \frac{1}{2} |L_i| \right\rfloor$ and the order is preserved.
   (b) For $k \in \{1, 2\}$, define the variance of L_i^k as

   $$v_{L_i^k} = \widetilde{w}_{L_i^k}^t V_{L_i^k} \widetilde{w}_{L_i^k},$$

   where $V_{L_i^k}$ is the covariance matrix between the constituents of L_i^k and

   $$\widetilde{w}_{L_i^k} = \frac{\operatorname{diag}\left[V_{L_i^k}\right]^{-1} 1_l}{1_l^T \operatorname{diag}\left[V_{L_i^k}\right]^{-1} 1_l}$$

   for $l = \left|L_i^k\right|$.
   (c) Compute the split factor

   $$\alpha_i = 1 - \frac{v_{L_i^1}}{v_{L_i^1} + v_{L_i^2}} \in [0, 1].$$

   (d) Re-scale allocations w_j by a factor of α_i for all $j \in L_i^1$.
   (e) Re-scale allocations w_j by a factor of $(1 - \alpha_i)$ for all $j \in L_i^2$.
4. Update the list of items. For every element $L_i \in L$ that has been bisected, replace L_i with its smaller subsets L_i^1 and L_i^2.
5. Loop to step 2

Recursive Bisection - Example

To demonstrate the recursive bisection, the example that has been elaborated in the hierarchical clustering step is resumed. In this example, a correlation matrix was given:

$$\rho = \begin{bmatrix} 1 & -0.3 & 0.5 & -0.4 \\ -0.3 & 1 & 0.1 & 0.8 \\ 0.5 & 0.1 & 1 & 0 \\ -0.4 & 0.8 & 0 & 1 \end{bmatrix}.$$

This matrix was then used to form clusters and to rearrange the assets:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} 2 \\ 4 \\ 1 \\ 3 \end{bmatrix}.$$

Suppose the corresponding standard deviations for one day returns were

$$\sigma_1 = 0.03, \quad \sigma_2 = 0.02, \quad \sigma_3 = 0.02, \quad \sigma_4 = 0.03.$$

Intuitively, the assets 2 and 3 are the least volatile, so they should receive more weight than the other assets if the goal of the asset allocation is to create a stable portfolio. However, as it has been shown before, the assets 2 and 4 form a close cluster. Thus, it might be reasonable not to give asset 2 too much weight, because otherwise a large part of the portfolio is invested in two assets that correlate strongly.
Given a correlation matrix and standard deviations, the corresponding covariance matrix can be derived:

$$\begin{aligned} V &= \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 \\ 0 & 0 & 0 & \sigma_4 \end{bmatrix} \rho \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 \\ 0 & 0 & 0 & \sigma_4 \end{bmatrix} \\ &= \begin{bmatrix} 0.0009 & -0.00018 & 0.0003 & -0.00036 \\ -0.00018 & 0.0004 & 0.00004 & 0.00048 \\ 0.0003 & 0.00004 & 0.0004 & 0 \\ -0.00036 & 0.00048 & 0 & 0.0009 \end{bmatrix} \\ &= 10^{-4} \begin{bmatrix} 9 & -1.8 & 3 & -3.6 \\ -1.8 & 4 & 0.4 & 4.8 \\ 3 & 0.4 & 4 & 0 \\ -3.6 & 4.8 & 0 & 9 \end{bmatrix}. \end{aligned}$$

Using the new order of assets, the covariance matrix is rearranged:

$$V = 10^{-4} \begin{bmatrix} 4 & 4.8 & -1.8 & 0.4 \\ 4.8 & 9 & -3.6 & 0 \\ -1.8 & -3.6 & 9 & 3 \\ 0.4 & 0 & 3 & 4 \end{bmatrix}.$$

Step 1: Initialize the algorithm:

(a) The first list of assets is set as $L = \{L_0\}$ with $L_0 = \{2, 4, 1, 3\}$.

(b) Unit weights are initialized: $w_2^{(0)} = w_4^{(0)} = w_1^{(0)} = w_3^{(0)} = 1$.

Step 2: Obviously, $|L_0| > 1$, so the algorithm continues.

Step 3: Select $L_0 = \{2, 4, 1, 3\}$.

(a) Bisect L_0 into $L_0^1 = \{2, 4\}$ and $L_0^2 = \{1, 3\}$.

(b) For L_0^1, the corresponding part of the covariance matrix is

$$V_{L_0^1} = 10^{-4} \begin{bmatrix} 4 & 4.8 \\ 4.8 & 9 \end{bmatrix}.$$

Therefore,

$$\operatorname{diag}\left[V_{L_0^1}\right]^{-1} = 10^4 \begin{bmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{9} \end{bmatrix}.$$

Thus, the inverse-variance allocation weights can be calculated as

$$\widetilde{w}_{L_0^1} = \frac{\operatorname{diag}\left[V_{L_0^1}\right]^{-1} 1_2}{1_2^T \operatorname{diag}\left[V_{L_0^1}\right]^{-1} 1_2} = \frac{10^4}{10^4\left(\frac{1}{4} + \frac{1}{9}\right)} \begin{bmatrix} \frac{1}{4} \\ \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{9}{13} \\ \frac{4}{13} \end{bmatrix}.$$

Finally, the variance of L_0^1 can be calculated as

$$v_{L_0^1} = \widetilde{w}_{L_0^1}^t V_{L_0^1} \widetilde{w}_{L_0^1} = 4,8 \cdot 10^{-4}.$$

For L_0^2, the variance is derived similarly as

$$v_{L_0^2} = \widetilde{w}_{L_0^2}^t V_{L_0^2} \widetilde{w}_{L_0^2} = 4 \cdot 10^{-4}.$$

(c) The split factor is computed by

$$\alpha_0 = 1 - \frac{v_{L_0^1}}{v_{L_0^1} + v_{L_0^2}} = 0.45673.$$

(d) The weights for the assets in L_0^1 are re-scaled:

$$w_2^{(1)} = w_4^{(1)} = \alpha_0 w_2^{(0)} = 0.45673.$$

(e) The weights for the assets in L_0^2 are re-scaled:

$$w_1^{(1)} = w_3^{(1)} = (1 - \alpha_0) w_1^{(0)} = 0.54327.$$

Step 4: Update the list of items with $L_1 = L_0^1$ and $L_2 = L_0^2$:

$$L = \{L_0\} \quad \rightarrow \quad L = \{L_1, L_2\}.$$

Step 5: Loop to step 2.

Step 2′: As $|L_1| > 1$ and $|L_2| > 1$, the algorithm continues.

Step 3′: Select $L_1 = \{2, 4\}$.

(a) Bisect L_1 into $L_1^1 = \{2\}$ and $L_1^2 = \{4\}$.

(b) As L_1^1 only contains one asset, $V_{L_1^1} = 4 \cdot 10^{-4}$, $\widetilde{w}_{L_1^1} = 1$ and $v_{L_1^1} = 4 \cdot 10^{-4}$. Similarly, it can be shown that $v_{L_1^2} = 9 \cdot 10^{-4}$.

(c) The split factor is computed by

$$\alpha_1 = 1 - \frac{v_{L_1^1}}{v_{L_1^1} + v_{L_1^2}} = 0.69231.$$

(d) The weights for the assets in L_1^1 are re-scaled:

$$w_2^{(2)} = \alpha_1 w_2^{(1)} = 0.3162.$$

(e) The weights for the assets in L_1^2 are re-scaled:

$$w_4^{(2)} = (1 - \alpha_1) w_4^{(1)} = 0.14053.$$

Select $L_2 = \{1, 3\}$.

(a) Bisect L_2 into $L_2^1 = \{1\}$ and $L_2^2 = \{3\}$.

(b) Compute $v_{L_2^1} = 9 \cdot 10^{-4}$ and $v_{L_2^2} = 4 \cdot 10^{-4}$.

(c) Compute the split factor: $\alpha_2 = 0.30769$.

(d) Re-scale the weights in L_2^1:

$$w_1^{(2)} = \alpha_2 w_1^{(1)} = 0.16716.$$

(e) Re-scale the weights in L_2^2:

$$w_3^{(2)} = (1 - \alpha_2) w_3^{(1)} = 0.37611.$$

Step 4′: Update the list of items with $L_3 = L_1^1$, $L_4 = L_1^2$, $L_5 = L_2^1$ and $L_6 = L_2^2$:

$$L = \{L_1, L_2\} \quad \rightarrow \quad L = \{L_3, L_4, L_5, L_6\}.$$

Step 5′: Loop to step 2.

Step 2″: $|L_i| = 1$ for all $L_i \in L$, so stop.

The final weights can be seen in Table 5.1. As suspected, the least volatile assets 2 and 3 receive the most weight. Out of the two clusters $\{2, 4\}$ and $\{1, 3\}$, the latter is favoured. This is not due to the individual asset's volatility, as both clusters have an asset with a standard deviation of 0.02 and 0.03, but because of their correlation structure. Assets 2 and 4 correlate strongly, therefore their combined weight is diminished.

Asset	1	2	3	4
Weight	0.16716	0.3162	0.37611	0.14053

Table 5.1: The final weights of the example that has been discussed.

The main advantage of the Hierarchical Risk Parity method is its omission of the inversion of the covariance matrix, which is why it is a very robust algorithm. In fact, the covariance matrix can be ill-conditioned or even singular. Moreover, the algorithm is fast, as it converges in logarithmic (best case) or linear (worst case) time, see [López, 2018, p. 238].
In this thesis, one variant of the HRP algorithm is introduced. Of course, it can be adjusted and improved. In the hierarchical clustering step, other distance measures d, $\widetilde{d}$ and $\mathring{d}$ can be used to form different clusters. It is even possible to use completely different clustering algorithms, as long as assets in clusters are placed together to form a quasi-diagonal covariance matrix. For a comparison of different clustering algorithms in finance, see [Kolanovic & Krishnamachari, 2017, pp. 93-98]. Furthermore, in the hierarchical clustering step, different functions for the variance v_L of a subset L and for the split factor α can be applied. Instead of carrying out the recursive bisection, the allocations could also be split following the hierarchical clusters from top to bottom as seen in a dendogram.
Empirical studies suggest that "hierarchical clustering based portfolios are robust, truly diversified and achieve statistically better riskadjusted performances than commonly used portfolio optimization techniques", see [Raffinot, 2018, p. 97]. The results in the case study in Chapter 6 also seem to be promising.

5.4 Post-Processing

When a trading strategy is tested with a backtest, an equity curve is generated. If the strategy went live, it would be closely monitored by the investor in order to be able to interfere when unforeseen price drops occur. Similarly, an arbitrary equity curve can be monitored automatically with post-processing.
The idea is to apply long-term trend indicators, which will be introduced in detail in Subsection 6.1.1, to the equity curve. Basically, the trend indicators display a positive trend '1' when the portfolio has been doing well. Price drops of the equity curve trigger the output of a negative trend '0'. This functions as a safeguard mechanism, as the trading strategy is only allowed to invest in assets when the underlying trend is positive, effectively overriding trading decisions of the trading strategy. If a negative trend is triggered, all assets are sold at the next day. Note that the asset selection and the asset allocation process is unaffected from this. After all assets have been sold because the trend turned negative, the trading strategy switches to simulating how its decisions would have played out. Basically, the portfolio is still monitored and reallocated, but trading decisions are not carried out. When the simulated equity curve starts to rally again, the trend indicator switches to a positive trend and the trading strategy is allowed to invest in the assets once again.

Every trend indicator has different parameters that affect its behaviour. Post-processing can be diversified by applying multiple trend indicators to the equity curve at once, each one responsible for a small share of equity. For example, suppose ten different trend indicators are used during post-processing. If one trend indicator switches to a negative trend, only a tenth of the equity is frozen and the other nine tenth are still invested, until the corresponding trend indicators turn negative as well.
As the trend indicators only need access to the past equity curve, post-processing can be integrated in the backtesting process, thus ensuring that a similar strategy could be applied on live data. The effect of post-processing will be analysed with real data in Subsection 6.3.3. For more details, see [Maier-Paape, 2016, p. 1-22].

Chapter 6

Case Study

In this chapter, the machine learning techniques are tested on actual data. Step by step, the chapter shows how to process the data, how to build machine learning models and how to apply them.
Underlying are two datasets: assets that make up the S&P 500 index from January 1990 to June 2020 and assets that make up the STOXX Europe 600 from January 2002 to June 2020. The datasets are survivorship bias free, so shares of companies that do not exist any more due to mergers or bankruptcy are included. Furthermore, assets are first taken into account when they enter the index. This way, no future knowledge of an asset entering the index is implied. Moreover, assets with a close series that spans less than 400 trading days are removed from the dataset. As some features have a backward-looking window of 250 trading days and one label a forward-looking horizon of 120 trading days, at least 92.5% of these asset's dataset would be discarded anyway. Furthermore, penny stocks with an adjusted close price of less than 5$ are removed, as small absolute price changes near zero result in massive return rates.

6.1 Data Processing

In Chapter 2, the structure of the underlying dataset as a matrix has been explained in detail. An extract of this dataset can be seen in Figure 6.1. A row of this matrix represents a sample, which describes the characteristics of a financial instrument at a point in time. In the columns, features and labels are displayed.
If a value cannot be calculated, as the underlying data is missing, it is displayed as NaN (Not a Number). For instance, the features of the first samples and the labels of the last samples are missing. When a machine learning model is fitted on a dataset, samples with missing feature values are removed.
The core of the data is a set of adjusted close stock prices. The most basic dataset consists of samples that have a date ('Date'), an asset symbol to identify the individual assets ('AssetSymbol'), the adjusted close price ('CloseAdj') and labels. Unfortunately, the feature 'AssetSymbol' is not numeric, so it cannot be interpreted by a random forest classifier. Therefore, this feature is not used by the final machine learning model.

Date	AssetSymbol	CloseAdj	Return10	...	Label90	Label120
				...		
1990-01-02	764144Q_US	2.9375	NaN	...	-1.0	-1.0
1990-01-02	2245Q_US	34.7500	NaN	...	-1.0	-1.0
1990-01-02	1778808D_US	0.7094	NaN	...	-1.0	1.0
1990-01-02	KMRTQ_US	18.3125	NaN	...	-1.0	-1.0
1990-01-02	0202445Q_US	12.1585	NaN	...	-1.0	1.0
...	...	...	...	...	...	...
2020-06-19	NI_US	22.9700	-0.084131	...	NaN	NaN
2020-06-19	IPGP_US	162.2400	-0.042662	...	NaN	NaN
2020-06-19	BA_US	187.0200	-0.089617	...	NaN	NaN
2020-06-19	USB_US	38.8700	-0.082606	...	NaN	NaN
2020-06-19	ZTS_US	137.4000	-0.025601	...	NaN	NaN

[5933976 rows x 179 columns]

Figure 6.1: An extract of the S&P 500 dataset.

As seen in Section 2.2, labels can be derived from the close price alone. In this example, three labels are added: the Edge Ratio label with a labelling horizon of 60, 90 and 120 trading days ('Label60', 'Label90' and 'Label120'). When a classifier is trained, it has access to one of these labels. This helps to diversify classifiers in the machine learning trading strategy, which has been mentioned in Subsection 5.2.3.
As explained in Chapter 4, machine learning models need access to meaningful features to give accurate predictions. These features have to be constructed and added to the data matrix. For example, in Figure 6.1 the 10-day return has been added as a feature 'Return10'. In Subsection 6.1.1, multiple features are introduced. In Subsection 6.1.2, they are analysed in order to select meaningful features.

6.1.1 Feature Engineering

There are countless possibilities to construct new features. The features used in this thesis are defined in this subsection. In general, they can be grouped together in regards to the data they are based on.

Time-Based

Every sample has a time stamp. Therefore, it is natural to analyse whether the time in itself has an influence on the price development.
The first time-based feature to analyse is the year of a sample ('Year'). Intuitively, it is not clear whether this is a good feature. On the one hand, when the classifier tries to predict the label of a current sample, there are few samples with a similar year in the training dataset. On the other hand, the year might help to differentiate between old and new data in the training dataset. Like all other features, the feature importance will show whether to include this feature in the final model.
Another interesting approach is to analyse whether the financial market follows seasonal trends. Rather than specifying the month of a sample, the day of the year ('DayOfYear')

is chosen to indicate the position of a sample in the respective year, because samples can be differentiated more precisely. The next interesting period of time to investigate is a month ('DayOfMonth'), as traders might be more anxious towards the end of the month. Finally, the weekday can be analysed as well ('DayOfWeek').

Index-Based

It would be a mistake to think an asset is independent from other assets. When most of the financial market is bullish, it is more likely that the price of an individual asset rises as well. Likewise, if the market is bearish, the risk of falling prices for any single asset is considerable. There are many ways to take into account the market development. As all assets in the datasets are part of either the S&P 500 or the STOXX Europe 600, the respective index can be used as a basis for calculations.
Based on all assets in the index, a price is formed. For $t \in \{1, \dots, T\}$, let p_t denote the close price of the respective index at day t. There are no splits or dividends, so adjusting the close price has no effect. To stay consistent with individual assets, the close price of the index is called adjusted close nonetheless:

$$(\text{IndexCloseAdj})_t = p_t,$$

for $t \in \{1, \dots, T\}$.
However, an isolated close price at a point in time does not reflect temporal developments very well. In order to give a sample information about the past, features have to be based on multiple data. The easiest way to do this is to calculate the return. The k-day return at day $t \in \{k+1, \dots, T\}$ can be calculated as

$$(\text{IndexReturn } k)_t = \frac{p_t}{p_{t-k}} - 1.$$

For $t \in \{1, \dots, k\}$, the k-day return cannot be calculated. This is denoted by NaN in the dataset. Similarly, if a value for any other feature cannot be calculated, an entry of NaN is placed in the dataset instead.
To indicate the window of past trading days a feature is based on, a number is added as a suffix. For instance, the 120-day return of the index is denoted as 'IndexReturn120'. If another feature has access to past data, a corresponding suffix is added as well. In Subsection 6.1.2, the optimal windows for individual features are studied.
Furthermore, the index' volatility can be analysed as well. In order to do this, the daily logarithmic returns of the close prices are calculated:

$$\widetilde{r}_t = \log\left(\frac{p_t}{p_{t-1}}\right).$$

A k-day moving average, here applied on the logarithmic returns, is defined as

$$\bar{\widetilde{r}}_t^{[k]} = \frac{1}{k}\sum_{i=0}^{k-1} \widetilde{r}_{t-i}.$$

As described in Subsection 2.2.1, the k-day rolling standard deviation can be estimated:

$$\sigma_{\widetilde{r},t}^{[k]} = \sqrt{\frac{1}{k-1}\sum_{i=0}^{k-1}\left(\widetilde{r}_{t-i} - \overline{\widetilde{r}}_t^{[k]}\right)^2}. \tag{6.1}$$

Thus, the volatility based on these daily logarithmic returns at $t \in \{k, \dots, T\}$ can be calculated by

$$(\text{IndexVola } k)_t = \sigma_{\widetilde{r},t}^{[k]}.$$

Instead of calculating the volatility by oneself, external data can be utilized. For instance, the Volatility Index (VIX) is a measure of the volatility of the stock market based on options of stocks related to the S&P 500, see [Appel, 2005, p. 154]. Before 2003, calculations were based on the smaller S&P 100 index. As the old and the new VIX are comparable, they are both listed as a feature 'VIX' in this thesis.
Finally, the drawdown of the close prices can be calculated as well. It is highly correlated to the volatility, which will be shown in Subsection 6.1.2. First, the rolling minimum and maximum for k days are defined:

$$p_{\min,t}^{[k]} = \min\{p_{t-k+1}, \dots, p_t\} \quad \text{and} \quad p_{\max,t}^{[k]} = \max\{p_{t-k+1}, \dots, p_t\}. \tag{6.2}$$

The drawdown with a rolling window of k trading days at $t \in \{k, \dots, T\}$ is then defined as

$$(\text{IndexDD } k)_t = \frac{p_{\max,t}^{[k]} - p_t}{p_{\max,t}^{[k]}} \in [0, 1].$$

Index-Based, Trend Indicators

When a feature is defined, it is essential that the feature gives some sort of temporal context, as the classifier is given one sample at a time. The index' close price at t is only a snapshot of the price series and the k-day return only takes into account two data points: the current close and the close k trading days prior. Therefore, indicators that better describe the temporal context of the close price have to be constructed. There are five trend indicators that are introduced in this thesis: an envelope around the moving average of the price, a Bollinger Band, a Donchian channel, the MACD and a trend oscillator. For more details, see [Maier-Paape, 2016, p. 4-10].
First of all, the close can be normalized. As the underlying data can be interpreted as a time series, this method can be visualized as an envelope around the graph of the moving average close price, as seen in Figure 6.2. The envelope has a fixed distance from the average close, for example $\pm 3\%$ of the average close, and the indicator describes the position of the close price relative to the envelope. The moving average close price at t based on the last k trading days can be calculated as

$$\overline{p}_t^{[k]} = \frac{1}{k}\sum_{i=0}^{k-1} p_{t-i}. \tag{6.3}$$

Thus, the relative position to the mean for an envelope of 3% at $t \in \{k, \dots, T\}$ is defined as

$$(\text{IndexEnvelope } k)_t = \frac{p_t - \overline{p}_t^{[k]}}{0.03\ \overline{p}_t^{[k]}}. \tag{6.4}$$

Figure 6.2: 3% envelope around the 150-day moving average of the adjusted close price.

Using an envelope, the close becomes more comparable to close prices at other points in time. The threshold, which is fixed as 3% in this thesis, helps to interpret the feature. A value of 1 indicates that the current close price is 3% higher than the moving average, as

$$\frac{p_t - \overline{p}_t^{[k]}}{0.03\ \overline{p}_t^{[k]}} \geq 1 \quad \Leftrightarrow \quad p_t \geq 1.03\ \overline{p}_t^{[k]}.$$

Similarly, a value of -1 indicates that the current price is 3% lower than the moving average. To concentrate on the essentials, a simpler feature can be constructed for $t \in \{k+1, \dots, T\}$:

$$(\text{IndexEvelopeTrend } k)_t = \begin{cases} 0 & \text{if } (\text{IndexEnvelope } k)_t \leq -1 \\ 1 & \text{if } (\text{IndexEnvelope } k)_t \geq 1 \\ (\text{IndexEvelopeTrend } k)_{t-1} & \text{else} \end{cases}$$

with the initialization

$$(\text{IndexEnvelopeTrend } k)_k = \begin{cases} 0 & \text{if } (\text{IndexEnvelope } k)_k \leq 0 \\ 1 & \text{if } (\text{IndexEnvelope } k)_k > 0 \end{cases}$$

Other initialization methods are possible. In general, it is desirable to start a feature as early as possible, as a NaN value at the beginning of the dataset cannot be interpreted by a classifier.

The 'Trend' feature has the added benefit that trends have more longevity in comparison to (6.4), as a trend changes its sign only when the respective opposite threshold of 3% or -3% is hit. To indicate that a trend indicator feature is simplified to a discrete set of values, the suffix 'Trend' is added.
Unfortunately, the envelope does not take into account the volatility of the close prices, which can change drastically over time. If the envelope around the moving average close price is based on the standard deviation of prices, it is called a Bollinger Band, see [Bollinger, 2002, xxii]. The rolling standard deviation of the price series $\sigma_{p,t}^{[k]}$ can be calculated by

$$\sigma_{p,t}^{[k]} = \sqrt{\frac{1}{k-1}\sum_{i=0}^{k-1}\left(p_{t-i} - \overline{p}_t^{[k]}\right)^2}. \tag{6.5}$$

The position of the close price relative to the Bollinger Bands at $t \in \{k, \ldots, T\}$ can be defined as

$$(\text{IndexBollinger } k)_t = \frac{p_t - \overline{p}_t^{[k]}}{2\,\sigma_{p,t}^{[k]}}.$$

Again, other values for the threshold are possible. In this thesis, it is fixed on the standard $2\,\sigma_{p,t}^{[k]}$.

Figure 6.3: A 2σ Bollinger Band around the 150-day moving average of the adjusted close price.

A simplified version of this feature can be constructed for $t \in \{k+1, \ldots, T\}$:

$$(\text{IndexBollingerTrend } k)_t = \begin{cases} 0 & \text{if } (\text{IndexBollinger } k)_t \leq -1 \\ 1 & \text{if } (\text{IndexBollinger } k)_t \geq 1 \\ (\text{IndexBollingerTrend } k)_{t-1} & \text{else} \end{cases}$$

with the initialization

$$(\text{IndexBollingerTrend } k)_k = \begin{cases} 0 & \text{if } (\text{IndexBollinger } k)_k \leq 0 \\ 1 & \text{if } (\text{IndexBollinger } k)_k > 0 \end{cases}$$

Another way to put the close price into a temporal context is to use Donchian channels, see [Bollinger, 2002, p. 40]. Basically, the indicator gives the position of the close price relative to a rolling minimum and maximum, which has been defined in (6.2). The position relative to the Donchian channel at $t \in \{k, \dots, T\}$, which can be seen in Figure 6.4, can be defined as

$$(\text{IndexDonchian } k)_t = \frac{p_t - p^{[k]}_{\min,t}}{p^{[k]}_{\max,t} - p^{[k]}_{\min,t}} \in [0,1].$$

Figure 6.4: A Donchian channel with a rolling window of 150 days around the adjusted close price.

In contrast to the envelope or the Bollinger Band, the current close price is always contained in the Donchian channel, as the channel is expanded when the current close becomes the new rolling minimum or maximum. The simplified trend at $t \in \{k+1, \dots, T\}$ is 0 or 1, depending on the last barrier the current close price has touched:

$$(\text{IndexDonchianTrend } k)_t = \begin{cases} 0 & \text{if } p_t = p^{[k]}_{\min,t} \\ 1 & \text{if } p_t = p^{[k]}_{\max,t} \\ (\text{IndexDonchianTrend } k)_{t-1} & \text{else} \end{cases}$$

with the initialization

$$(\text{IndexDonchianTrend } k)_k = \begin{cases} 0 & \text{if } p_k \leq \frac{1}{2}\left(p^{[k]}_{\min,k} + p^{[k]}_{\max,k}\right) \\ 1 & \text{else} \end{cases}$$

so the first value for the trend is chosen depending on whether the current price at $t = k$ is nearer to the rolling minimum or maximum.
The three trend indicators that have been introduced so far are all based on some sort of envelope or channel around the close price. There are other indicators that try to capture the momentum of a price.
For instance, the Moving Average Convergence-Divergence (MACD) indicator is constructed with moving averages with different time windows in order to detect the underlying trend, see [Appel, 2005, pp. 165-172]. The indicator uses exponential moving averages of the price series, which gives more weight to recent data points. The s-day exponential moving average at day $t \in \{2, \dots, T\}$ is defined as

$$\begin{aligned} \mathrm{EMA}_1(p, s) &= p_1 \\ \mathrm{EMA}_t(p, s) &= \frac{2}{s+1} p_t + \left(1 - \frac{2}{s+1}\right) \mathrm{EMA}_{t-1}(p, s). \end{aligned} \tag{6.6}$$

The quotient $\frac{2}{s+1}$ is typically used to calculate the s-day exponential moving average, see [Appel, 2005, pp. 134-135].
However, there is an issue with this definition when used to construct features. According to (6.6), even short-term exponential moving averages have access to the whole price series starting from $t = 1$. Thus, it is possible that a feature based on exponential moving averages has access to samples in the test dataset, if it predates the training dataset. As explained in Subsection 3.1.3, this happens when using cross-validation. In order to keep track of the number of days k a feature has access to, k is always displayed in the feature name.
A possible workaround would be to apply (6.6) with a rolling window by moving the start of the exponential moving average a day at a time. Unfortunately, this solution has two key problems.
Firstly, the beginning of the exponential moving average series in (6.6) greatly depends on the starting value. If the underlying price series is volatile, starting the calculation a day later can have a big impact on the exponential moving average. As the moving average becomes more volatile, so does the rolling feature. In practice, a feature at day t_0 might differ drastically from a feature at day $t_1 = t_0 + 1$, although the underlying price series has not changed apart from one value.
Secondly, (6.6) requires a recursive calculation. Applying a rolling window hence forces the computer to re-calculate the whole series for every trading day, which is very time consuming and therefore not feasible.
In order to circumvent these problems, an alternative definition of an exponential moving average with a window of k trading days at day $t \in \{k, \dots, T\}$ is used in this thesis:

$$\mathrm{EMA}_t^{[k]}(p, s) = \frac{\sum_{i=0}^{k-1} w_i p_{t-i}}{\sum_{i=0}^{k-1} w_i}, \tag{6.7}$$

with

$$w_i = \left(1 - \frac{2}{s+1}\right)^i$$

for $i \in \{0, \ldots, k-1\}$.
For an infinite underlying price series, both definitions are identical. In order to see this, consider

$$\mathrm{EMA}_t^{[\infty]}(p,s) = \frac{p_t + \left(1 - \frac{2}{s+1}\right) p_{t-1} + \left(1 - \frac{2}{s+1}\right)^2 p_{t-2} + \ldots}{1 + \left(1 - \frac{2}{s+1}\right) + \left(1 - \frac{2}{s+1}\right)^2 + \ldots}.$$

As the denominator is a geometric series with initial term equal to 1 and a ratio of $1 - \frac{2}{s+1}$, this can be rearranged:

$$\begin{aligned}
\mathrm{EMA}_t^{[\infty]}(p,s) &= \frac{p_t + \left(1 - \frac{2}{s+1}\right) p_{t-1} + \left(1 - \frac{2}{s+1}\right)^2 p_{t-2} + \ldots}{\frac{1}{1-\left(1-\frac{2}{s+1}\right)}} \\
&= \left(p_t + \left(1 - \frac{2}{s+1}\right) p_{t-1} + \left(1 - \frac{2}{s+1}\right)^2 p_{t-2} + \ldots\right) \frac{2}{s+1} \\
&= \frac{2}{s+1} p_t + \left(\left(1 - \frac{2}{s+1}\right) p_{t-1} + \left(1 - \frac{2}{s+1}\right)^2 p_{t-2} + \ldots\right) \frac{2}{s+1} \\
&= \frac{2}{s+1} p_t + \left(1 - \frac{2}{s+1}\right) \left(p_{t-1} + \left(1 - \frac{2}{s+1}\right) p_{t-2} + \ldots\right) \frac{2}{s+1} \\
&= \frac{2}{s+1} p_t + \left(1 - \frac{2}{s+1}\right) \mathrm{EMA}_{t-1}^{[\infty]}(p,s)
\end{aligned}$$

which is similar to (6.6).
Using a short warm-up, the beginning of the exponential moving average series is consistent in a rolling scheme. Furthermore, using vectorization, the calculation of (6.7) can be sped up drastically. For the implementation of an exponential moving average with a limited time window, see A.3 in the Appendix.
The original MACD is the difference between the 12-day exponential moving average and the 26-day exponential moving average:

$$\mathrm{MACD}_t = \mathrm{EMA}_t(p, 12) - \mathrm{EMA}_t(p, 26).$$

To be more sensitive to short-term price developments, an additional 9-day exponential moving average is added to the model. In the original trading strategy, when the MACD exceeds the additional signal line, a buy signal is generated.
In order to evaluate long-term developments, the moving averages have to span a bigger time period. To be consistent with the original MACD trading strategy, the relations between the moving averages stay the same. Hence, instead of 12, 26 and 9, the values $\frac{12}{26}k$, $\frac{26}{26}k$ and $\frac{9}{26}k$ are used to derive the exponential moving averages in relation to the rolling window k. The rolling k-day MACD can thus be defined as

$$\mathrm{MACD}_t^{[k]} = \mathrm{EMA}_t^{[k]}\left(p, \frac{6}{13}k\right) - \mathrm{EMA}_t^{[k]}\left(p, k\right).$$

In addition, the feature used in this thesis uses an envelope around the signal line based on the standard deviation of the MACD series. The final feature describes the position

of the MACD relative to the signal line, scaled by the standard deviation of the MACD line:

$$(\text{IndexMACD } k)_t = \frac{1}{\sigma^{[k]}_{\text{MACD},t}} \left(\text{MACD}^{[k]}_t - \text{EMA}^{[k]}_t \left(p, \frac{9}{26} k \right) \right),$$

where $\sigma^{[k]}_{\text{MACD},t}$ denotes the rolling standard deviation of the MACD at day t, which is calculated similar to (6.5). Thus, the value surpasses 1 if the MACD is greater than the signal line plus the standard deviation of the MACD. Similarly, it is smaller than -1 if the MACD signal is below the lower threshold.

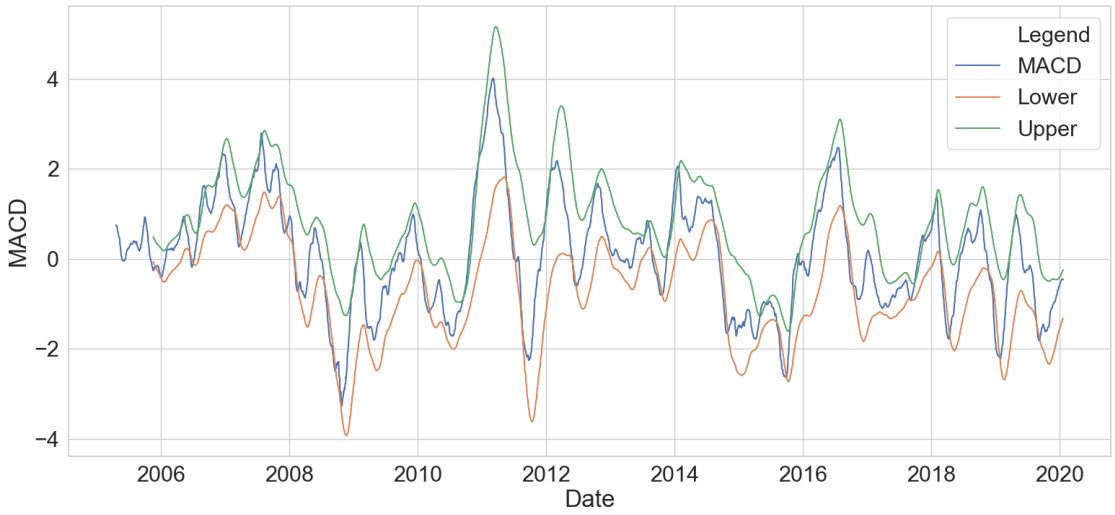

Figure 6.5: The 150-day MACD with an envelope around the signal line.

The simplified version of this feature can be constructed for $t \in \{k+1, \ldots, T\}$ as follows:

$$(\text{IndexMACDTrend } k)_t = \begin{cases} 0 & \text{if } (\text{IndexMACD } k)_t \leq -1 \\ 1 & \text{if } (\text{IndexMACD } k)_t \geq 1 \\ (\text{IndexMACDTrend } k)_{t-1} & \text{else} \end{cases}$$

with the initialization

$$(\text{IndexMACDTrend } k)_k = \begin{cases} 0 & \text{if } (\text{IndexMACD } k)_k \leq 0 \\ 1 & \text{if } (\text{IndexMACD } k)_k > 0 \end{cases}$$

Furthermore, a feature can be based on the trend oscillator by Prof. Dr. Stanislaus Maier-Paape, see [Maier-Paape, 2016, p. 9]. Similar to the MACD, the trend oscillator combines several moving averages. Note that the trend oscillator that is introduced in this thesis is a simplified version. Whereas the original uses a combination of both regular and exponential moving averages, the trend oscillator in this thesis relies on exponential moving averages alone. For a snippet of the simplified algorithm in pseudo code, see Appendix A.

Firstly, a 'gradient' of the exponential moving average of the price is calculated:

$$\text{grad}_t^{[k]} = \frac{\text{EMA}_t^{[k]}(p,k) - \text{EMA}_{t-1}^{[k]}(p,k)}{p_t}.$$

Secondly, a short-term exponential moving average of this gradient is calculated and transformed so that the values range from 0 to 1:

$$\text{osci}_t^{[k]} = \frac{1}{2} + \frac{1}{\pi}\arctan\left(200 \cdot \text{EMA}_t^{[k]}\left(\text{grad}^{[k]}, 11\right)\right) \in [0,1].$$

This signal is called the oscillator. In this thesis, a 11-day exponential moving average is used, although other time spans are possible.
The value of the oscillator can vary a lot when the underlying prices are volatile. Therefore, an envelope around $\frac{1}{2}$ is added to account for the volatility. In order to do that, the volatility is derived from the osci time series:

$$v_t = \left(\text{osci}_t^{[k]} - \frac{1}{2}\right)^2.$$

As described in (6.3), a rolling volatility $\overline{v}_t^{[k]}$ is calculated. This volatility is used to derive the lower and upper envelope:

$$\text{lower}_t^{[k]} = \frac{1}{2} - \frac{7}{10}\overline{v}_t^{[k]} \quad \text{and} \quad \text{upper}_t^{[k]} = \frac{1}{2} + \frac{7}{10}\overline{v}_t^{[k]}.$$

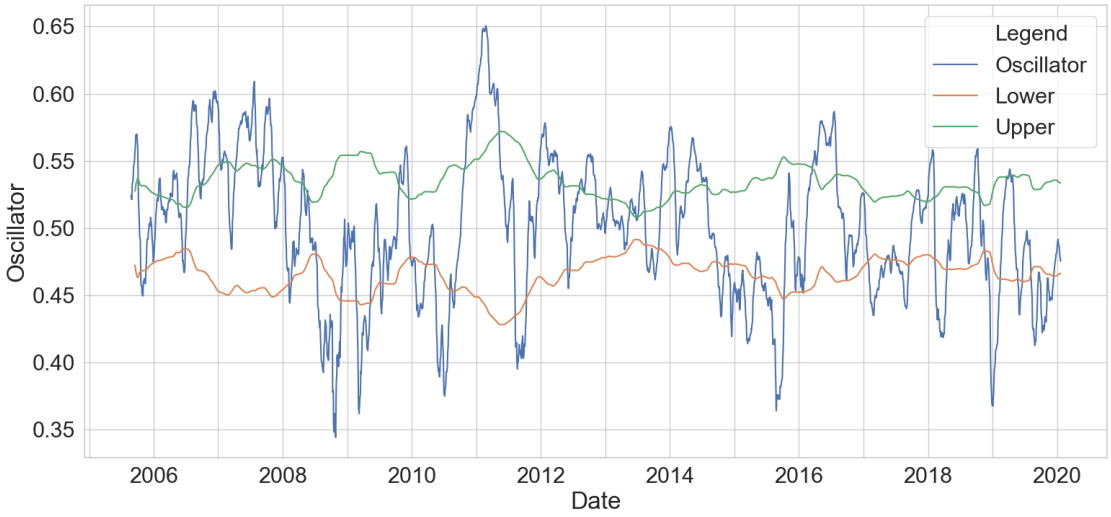

Figure 6.6: The 150-day trend oscillator with a volatility based envelope around 0.5.

The relative position of the oscillator can be added as a feature:

$$(\text{IndexOscillator } k)_t = \frac{\text{osci}_t^{[k]} - \frac{1}{2}}{\frac{7}{10}\overline{v}_t^{[k]}}.$$

Thus, the value for this feature is 1, iff the value of the oscillator is at the upper envelope. Similarly, it is -1, iff the oscillator hits the lower envelope. Again, other values for the volatility factor, which is chosen as $\frac{7}{10}$ in this thesis, are possible.
A simplified version of this feature can be constructed for $t \in \{k+1, \ldots, T\}$:

$$(\text{IndexOscillatorTrend } k)_t = \begin{cases} 0 & \text{if } (\text{IndexOscillator } k)_t \leq -1 \\ 1 & \text{if } (\text{IndexOscillator } k)_t \geq 1 \\ (\text{IndexOscillatorTrend } k)_{t-1} & \text{else} \end{cases}$$

with the initialization

$$(\text{IndexOscillatorTrend } k)_k = \begin{cases} 0 & \text{if } (\text{IndexOscillator } k)_k \leq 0 \\ 1 & \text{if } (\text{IndexOscillator } k)_k > 0 \end{cases}$$

Index-Based, Adjusted Trend Indicators

The trend indicators that have been introduced all make use of some sort of envelope. Inspired by the Donchian channel, all these indicators can be adjusted. Machine learning algorithms offer novel ways to analyse financial indicators. To test these methods, new versions of the previous mentioned indicators are added and analysed.
The main idea is simple: instead of using the normal lower and upper threshold of an indicator, rolling minima and maxima can be utilized. Suppose there is a signal s_t at $t \in \{1, \ldots, T\}$. Furthermore, there is an envelope ('lower', 'upper') that sets thresholds for this signal. Instead of using the original lower threshold for the signal, the adjusted lower threshold at $t \in \{k, \ldots, T\}$ is formed by using a rolling maximum $\left(\text{lower}^{[k]}_{\max,t}\right)$ as described in (6.2). Similarly, an adjusted upper threshold can be formed $\left(\text{upper}^{[k]}_{\min,t}\right)$. Hence, a positive trend is generated when the signal s_t exceeds $\text{upper}^{[k]}_{\min,t}$ rather than upper_t.

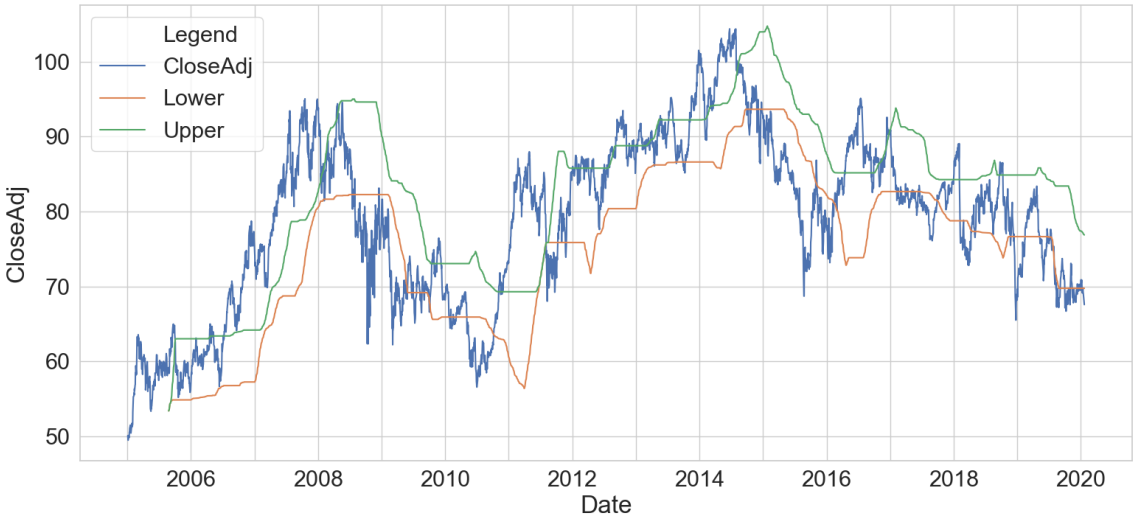

Figure 6.7: An adjusted 150-day Bollinger Band with a rolling maximum lower and a rolling minimum upper threshold ('IndexBollMaxMin150').

An example of this technique can be seen in Figure 6.7, where the original Bollinger Band from Figure 6.3 has been adjusted.
A rolling maximum of the lower threshold enables the indicator to detect downward trends more quickly, as the lower threshold does not drop with the current signal. For instance, the Bollinger Band might detect trends too late, as the lower threshold declines when the price drops, because the standard deviation rises sharply and the moving average price falls as well. Similarly, if prices are down, an upward trend can be detected faster when a rolling minimum of the upper threshold is used. This technique offers a new approach to established trend indicators.
In this thesis, there are three possible settings for a threshold: the original value can be used, a rolling minimum can be applied, or a rolling maximum can be utilized. As the lower and the upper threshold can be adjusted independently, this gives a total of nine different settings for an indicator.
To denote such an adjusted indicator, a suffix is added to show that a rolling minimum ('Min') or maximum ('Max') is applied, or that the original threshold is used and no function applied ('None'). The first suffix stands for the lower, the second for the upper threshold. Thus, 'IndexBollMinNone k' would indicate an adjusted k-day Bollinger Band, with a k-day rolling minimum lower and the original upper threshold. To limit the length of the feature name, abbreviations are used for the 'Envelope' ('Env'), 'Bollinger' ('Boll') and 'Oscillator' ('Osci') features when they are adjusted. The Donchian channel is already constructed with rolling minima and maxima, therefore the channel is not adjusted in this thesis.

Figure 6.8: An adjusted 150-day Bollinger Band with a rolling minimum lower and a rolling maximum upper threshold ('IndexBollMinMax150').

When the indicators are adjusted with rolling minima and maxima, it is possible that the lower threshold exceeds the upper threshold and vice versa. This would be problematic, as the signal could qualify for both a positive and a negative trend at the same time. To circumvent this, the upper threshold is bounded by the lower threshold, as seen in Figure 6.9. Therefore, instead of crossing, the lower and upper threshold would

merge. Of course, other techniques are possible. For example, a small envelope around the mean of the crossed lower and upper threshold could be constructed.

Figure 6.9: An adjusted 3 % envelope around a 150-day moving average of the close price with a rolling maximum lower and the original upper threshold ('IndexEnvMaxNone150').

Price-Based, Individual Asset

A sample in the dataset describes the characteristics of an individual asset at a given trading day. Of course, the market trend is a large factor to consider when giving predictions. However, characteristics that are unique to an asset can be used for predictions as well.
Most importantly, a stock has a price that can be used to construct features. As explained earlier with the index-based features, the adjusted close price ('CloseAdj') is used in this thesis. Again, the close has to be put in a temporal context. This can be achieved by applying the same features used for the index for the individual assets. To differentiate these features to the features based on the index, the prefix 'Index' is removed.
For instance, let $X_1 \in \mathbb{R}^{M+1}$ denote a sample at t_0 and $(p_t)_{t\in\{1,\dots,t_0\}}$ the price series of the underlying asset. The feature 'Return k' describes the k-day return of the underlying asset:

$$(\text{Return } k)_{t_0}(X_1) = \frac{p_{t_0}}{p_{t_0-k}} - 1.$$

For another sample $X_2 \in \mathbb{R}^{M+1}$ at t_0 that describes a different asset, the feature 'Return k' is calculated using the prices $(\widetilde{p}_t)_{t\in\{1,\dots,t_0\}}$ of the underlying asset of X_2:

$$(\text{Return } k)_{t_0}(X_2) = \frac{\widetilde{p}_{t_0}}{\widetilde{p}_{t_0-k}} - 1.$$

Therefore, the two samples X_1 and X_2 have different values for the feature 'Return k'. However, if both assets are listed in the same index, the value for 'IndexReturn k' is

the same, as both samples share the same date t_0. Thus, all features that have been introduced for a index can be applied for individual stock prices as well. Although the trend indicators have been designed to describe index prices, they might still be useful on the more volatile individual stock prices.
When designing good features, the feature correlation should be taken into account, as correlated features only describe the same characteristic multiple times. For example, the return of a stock price correlates with the return of the index price, which can be seen in Figure 6.10. Thus, rather than using the features 'Return k' and 'IndexReturn k', uncorrelated features are added. Two easy ways to adjust the stock price's return are used in this thesis. Firstly, the return of the index can be subtracted from the stock price return, creating a residualised return that solely depends on an asset's performance:

$$\text{ReturnIndexRes } k = \text{Return } k - \text{IndexReturn } k.$$

Secondly, the ratio of the returns can be build, as described in Subsection 3.3.1:

$$\text{ReturnIndexRatio } k = \frac{\text{Return } k}{\text{IndexReturn } k}.$$

A value for 'IndexReturn k' that is almost zero or negative might create a ratio that is hard to interpret. This will be analysed with the feature importance in Subsection 6.1.2.

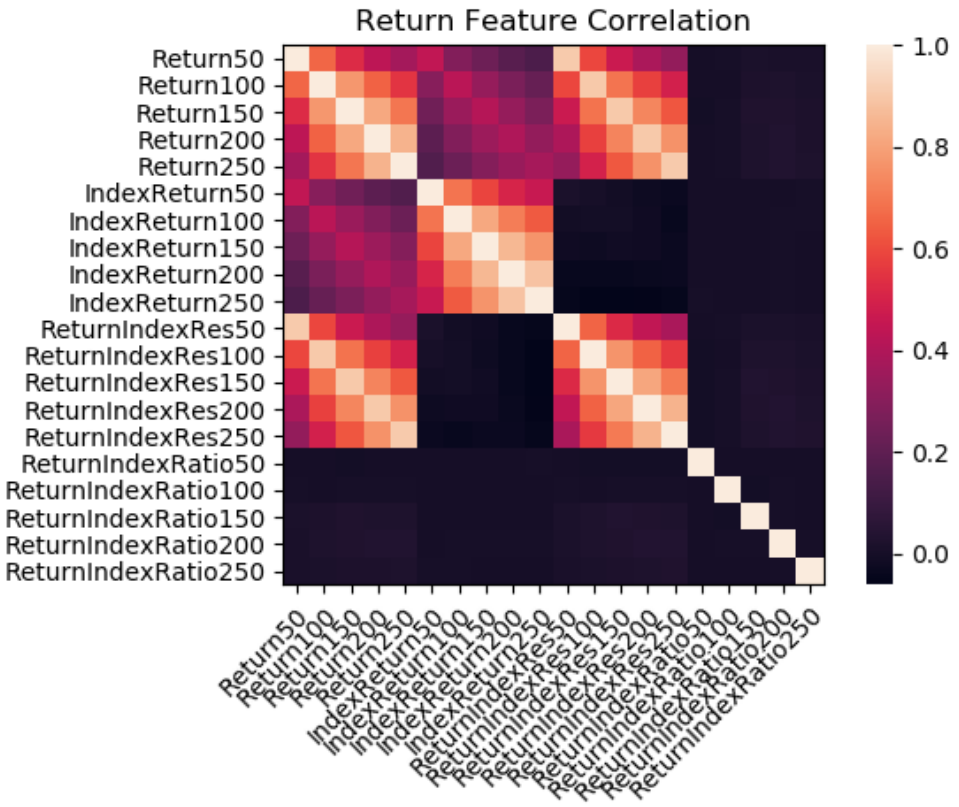

Figure 6.10: Feature correlation of return features for the S&P 500 dataset.

Price-Based, Market Structure

Instead of using the index as an indicator for the market movement, the constituents of the index can be analysed directly. This is done by aggregating the price-based features that have been introduced for the individual assets.
Let $\mathcal{X}_t$ denote the set of samples that describe assets of a given market at $t \in \{1, \dots, T\}$. In particular, $|\mathcal{X}_t|$ is the number of assets contained in that market at day t. One way to aggregate features is to compute the arithmetic mean. For example, the feature 'EnvelopeTrend k' can be transformed into a market feature:

$$(\text{MrktEnvelope } k)_t = \frac{1}{|\mathcal{X}_t|} \sum_{X \in \mathcal{X}_t} (\text{EnvelopeTrend } k)_t(X) \in [0, 1].$$

Features that have been calculated by that method are denoted with a prefix 'Mrkt'. In theory, any individual feature can be aggregated to form a market feature. However, using discrete features is preferable, as they are less prone to outliers and sudden market shifts become visible more quickly. Therefore, except for the features 'MrktVola k', 'MrktDD k' and 'MrktReturn k', all market features are based on their discrete counterparts that are denoted with a 'Trend'. For instance, the feature 'MrktDonchian k' is build on 'DonchianTrend k' rather than 'Donchian k'. As the 'Trend' features are either 0 or 1, the market feature can be interpreted as the proportion of assets that have a positive trend according to the underlying indicator.

6.1.2 Feature Selection

It is relatively easy to construct new features to add to the model, but it is hard to create meaningful features. Adding redundant or insignificant features might even impair a model. Therefore, it is a good idea to analyse the features in order to only select meaningful ones.
The following analysis of the feature importance is restricted to the years 1990-2004 (for the S&P 500 dataset) and 2003-2011 (for the STOXX Europe 600 dataset). The years after that are part of the test dataset, which will be used for the backtest in Section 6.3. Therefore, the classifier should not have access to these samples beforehand.

Window Length

Most of the features have a rolling window of past samples they use for calculations. In order to analyse the impact of the rolling window on a feature, the labelling horizon has to be taken into account. As described in Chapter 4, the feature importance indicates how much a feature contributes to successful predictions. Therefore, the feature importance is based on the label the classifier uses, as it determines how to classify samples. In Figures 6.11, 6.12 and 6.13, it is apparent that features with a long rolling window generally have a high feature importance. Most of the features with a 50-day window rank relatively low, even for a short labelling horizon of 60 days. Furthermore, some features have a high feature importance for every labelling horizon ('MrktVola250', 'MrktBollinger250', 'MrktDonchian250'). Others always have a low feature importance ('MrktEnvelope50', 'MrktReturn50', 'MrktDD50').

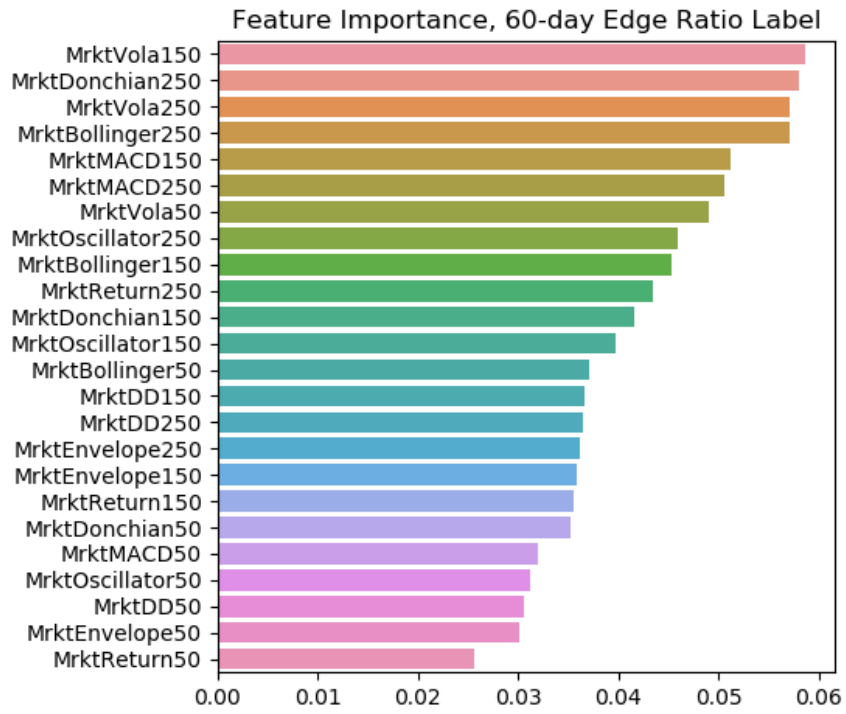

Figure 6.11: Feature importance for selected market features for the S&P 500 dataset.

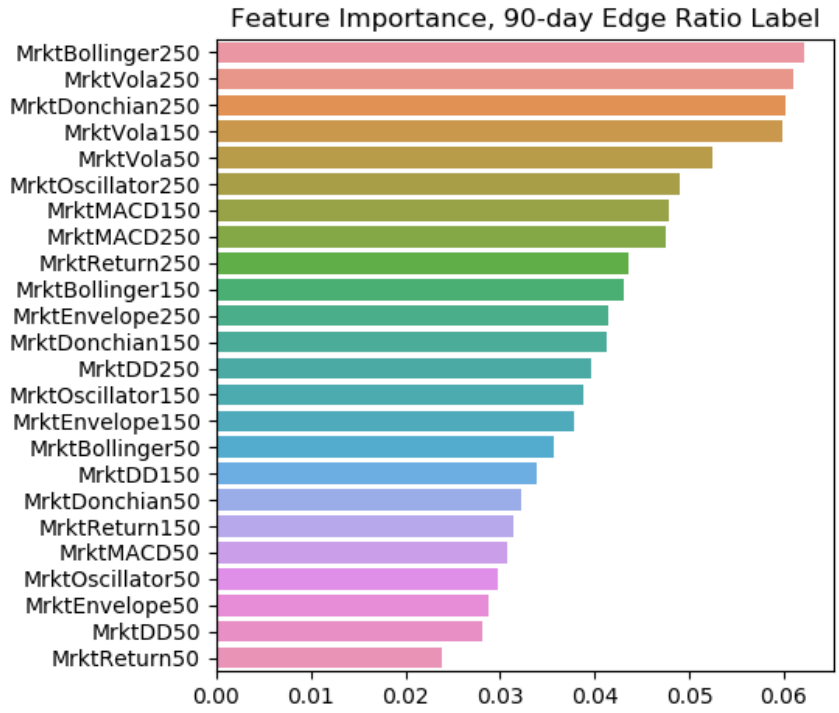

Figure 6.12: Feature importance for selected market features for the S&P 500 dataset.

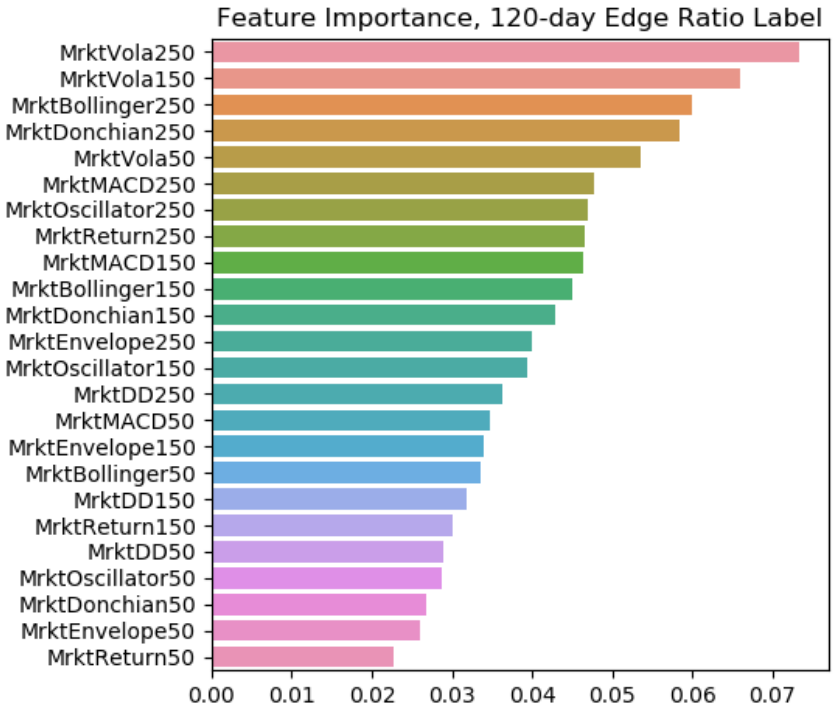

Figure 6.13: Feature importance for selected market features for the S&P 500 dataset.

It would be possible to use features with a large rolling window exclusively, as these features tend to have a high feature importance. However, in order to give a comprehensive description of the state of a sample, multiple different rolling windows should be used for a given feature. In this thesis, the features have rolling windows of 10, 25, 50, 100, 150, 200 and 250 trading days, thus giving the classifier the ability to react to both short-term and long-term trends.

Adjusted Trend Indicators

As explained in Subsection 6.1.1, trend indicators can be adjusted by using the rolling minimum or maximum of thresholds. In total, there are nine different combinations of rolling minima and maxima for the lower and the upper threshold.
The feature importance for the adjusted MACD feature is displayed in Figure 6.14. First of all, the distinction between the 'Index' and the 'Mrkt' features is obvious. As explained in Section 4.1, the mean decrease impurity (MDI) is used to measure the feature importance. Discrete features generally have a lower MDI, as a decision tree can split the dataset on this feature only once. This explains why the 'Trend' features have a relatively low feature importance. However, although 'IndexMACD150' is not a discrete feature, it still ranks lower than the other 'Mrkt' features. Therefore, Figure 6.5 seems to suggest that building MACD features based on all individual assets yields a higher feature importance than relying on the index price alone. This observation is consistent with the other trend indicators, which can be seen in Appendix B.

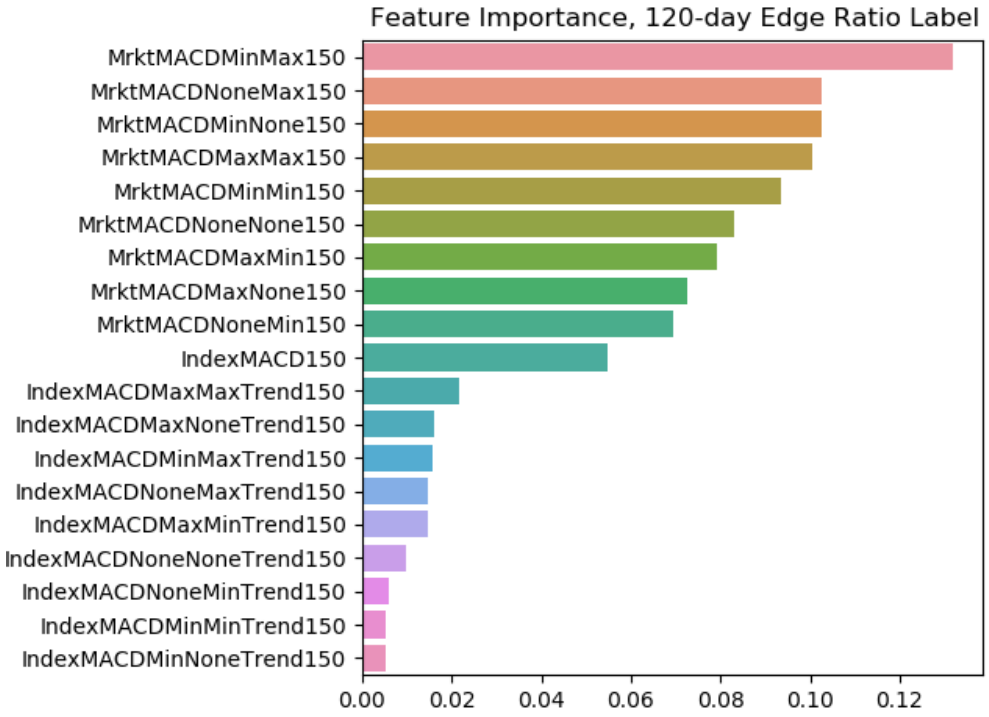

Figure 6.14: Feature importance for adjusted MACD features for the S&P 500 dataset.

Furthermore, the feature importance of trend indicators can be improved by using adjusted thresholds, since the original market indicator ('NoneNone') rarely ranks high. Surprisingly, the 'MinMax' configuration has the highest feature importance for most of the market indicators, which can be seen in Appendix B. In general, configurations that produce a wide envelope ('MinMax', 'NoneMax', 'MinNone') perform well. This could indicate that the width of the original envelope is set too tight in all market indicators. Alternatively, it might suggest that it is advantageous to stick to a trend when prices fall or rise, as the 'MinMax' envelope widens in these scenarios, which makes it harder to switch the label. When prices move horizontally for a long period of time, the 'MinMax' configuration produces a narrower envelope, which makes it easier to detect new trends. For 'Index' features, this observation is not that obvious, as features that detect a falling trend quickly ('MaxMax', 'MaxNone') have a higher feature importance compared to the other 'Trend' features. Nevertheless, the 'MinMax' configuration still ranks relatively high.

Using the 'MinMax' configuration to detect entry signals to buy stocks probably does not work well, as trends are detected very late, which can produce high costs if a stock is sold too late and high opportunity costs if a stock is bought too late. However, a classifier needs features to be accurate at any given trading day in order to produce accurate predictions. This shift of focus from finding the exact time a trend changes to accessing the current trend accurately might explain the high feature importance of the 'MinMax' configuration.

As the feature importance of the adjusted 'MinMax' indicators is generally higher than the feature importance of the original 'NoneNone' indicators, the final model uses market features with the 'MinMax' constellation.

Market Features

A look at all market features in Figure 6.15 shows similar results to Figure 6.14: 'Mrkt' features generally have a higher feature importance than 'Index' features and the 'MinMax' constellation is advantageous. Therefore, 'Mrkt' features are chosen over 'Index' features for the final model.

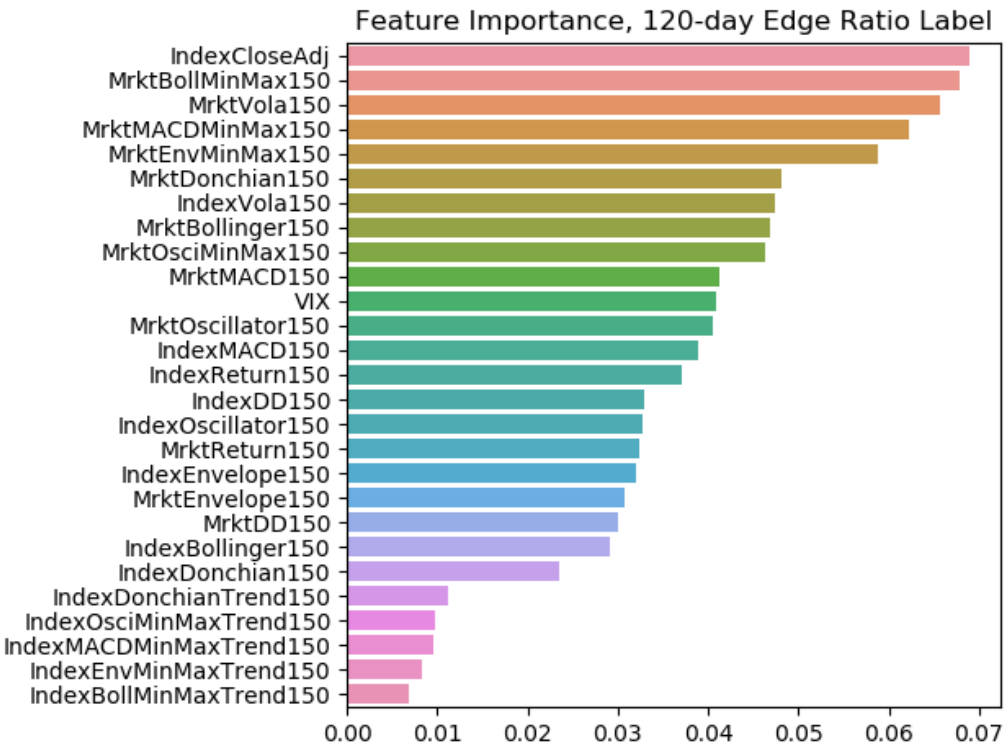

Figure 6.15: Feature importance for selected market features for the S&P 500 dataset.

Interestingly, the adjusted close of the index has a very high feature importance. Maybe the classifier is able to attribute samples to historic market phases due to the respective current index close. Furthermore, the MDI tends to be lower if there are multiple features that correlate highly, as two correlated features are to a certain degree interchangeable. The heatmap in Figure 6.16 shows that there are essentially three blocks of market features: trend indicators, features that describe the market volatility and the close of the index. That would explain why a unique feature like 'IndexCloseAdj' has a high feature importance. Moreover, the feature 'MrktBollMinMax150' seems to be somewhat independent of features like the market return, which boosts its feature importance. In general, unique and independent features are desirable, as highly correlated features explain the same characteristic of a sample and therefore might lead to a complex model with unnecessary variables that is prone to overfitting.

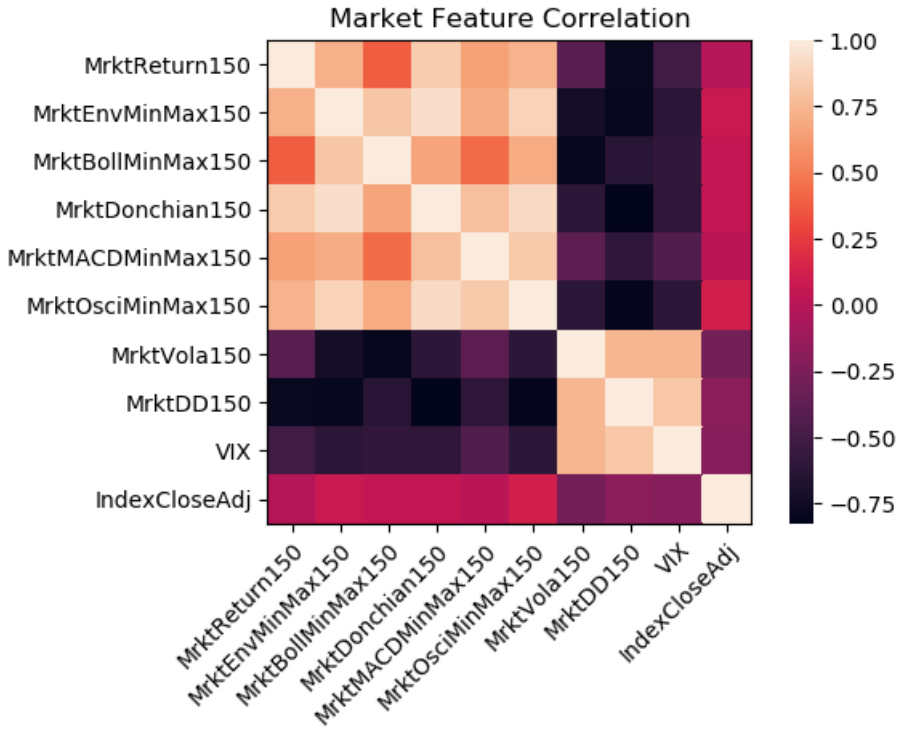

Figure 6.16: Feature correlation for selected market features for the S&P 500 dataset.

The feature importance for the STOXX Europe 600 dataset can be seen in Figure B.8 in the appendix. The feature 'VIX' has a relatively high feature importance, even though it describes the volatility of the US-American rather than the European market. Overall, the market feature importance plots for the STOXX Europe 600 and the S&P 500 datasets are comparable.

Individual and Time-Based Features

The feature importance of the rest of the features can be seen in Figure 6.17. Similar to the close of the index, the close of the individual stock seems to contain information as well. Maybe stocks with a high price have been successful historically, which would explain why they have a high price in the first place. Furthermore, the close price is uncorrelated to all other features, which elevates its feature importance.
The time-based features offer some surprises: The feature 'Year' has a very high feature importance, probably because it helps to relate a sample in the training set to the market phase of its respective time. For instance, samples in the year 1999 presumably behave differently from samples in the year 2008. When the label of a new sample is predicted, recent samples in the training dataset are given more relevance, as their year is similar to the new sample. Interestingly, the feature 'DayOfYear' ranks relatively high as well. Maybe there really are seasonal effects. However, the causation is not clear. If characteristic events like price rallies tend to happen in specific seasons, that does not mean that they are caused by the season. Nevertheless, the feature importance plot for the STOXX Europe 600 dataset, which stretches another time period, also shows a high

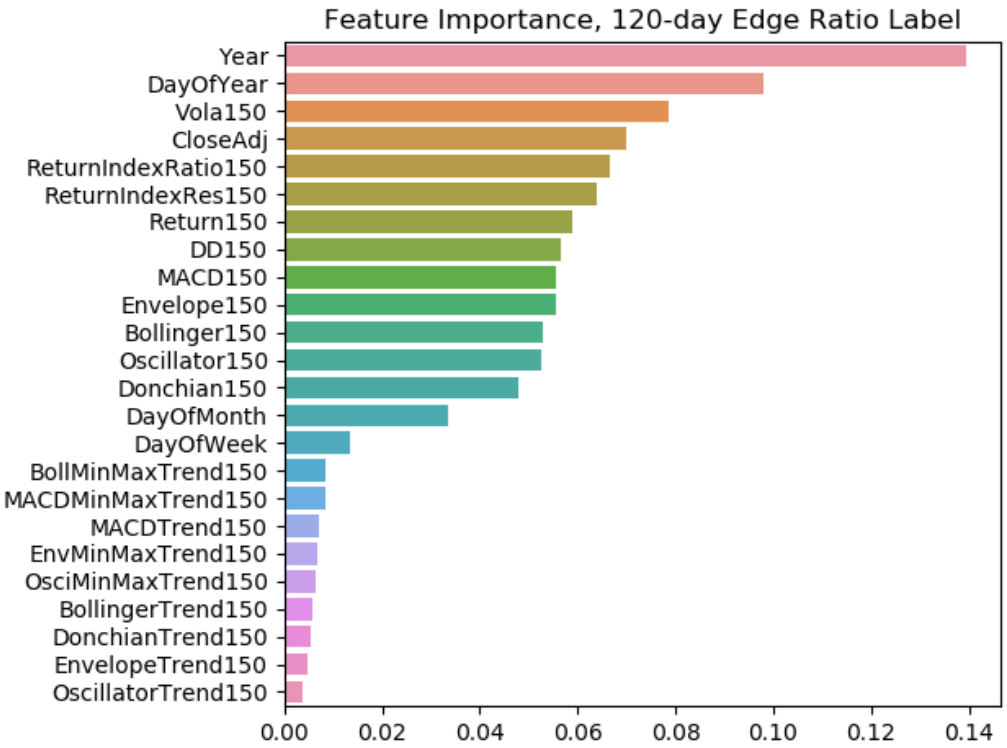

Figure 6.17: Feature importance for selected features for the S&P 500 dataset.

feature importance for 'DayOfYear', which can be seen in Figure B.9 in the appendix. In order to analyse seasonal effects further, more datasets that stretch over longer periods of time would have to be evaluated. The other two time-based features ('DayOfMonth' and 'DayOfWeek') have a relatively low feature importance and are thus not included in the list of features for the final model.
Similar to the market features, discrete 'Trend' features only have a small feature importance. In order to limit the number of unnecessary features, they are thus not used by the final classifier as well. The final list of features can be seen in Appendix C.

6.2 Hyperparameter Tuning

After meaningful features have been selected, a classifier has to be specified that can use these features. In Sections 3.3 and 3.4, two possible models have been introduced: decision trees and random forests.
As mentioned in Subsection 3.1.3, the dataset used for hyperparameter tuning is restricted to the years 1990-2004 (for the S&P 500 dataset) and 2003-2011 (for the STOXX Europe 600 dataset), as the other half of the respective dataset is used for a backtest in Section 6.3. Using samples from the test dataset would improve the performance of the final model. However, to simulate a realistic test scenario in the backtest and to reduce overfitting, only information that is available at the start of the backtest can be used in order to train the model.

6.2.1 Decision Tree

An extract of a hyperparameter tuning of a decision tree can be seen in Figures 6.18. In the hyperparameter tuning, different combinations of three parameters have been tested: the impurity measure ('criterion'), the maximal depth of the tree ('max_depth') and the number of features to choose from when splitting the dataset at a node ('max_features'). For all tests, a 120-day Edge Ratio label has been utilized. Of course, there are many hyperparameters that could have been tested additionally.

```
Testing parameter combination 1 of 18.
{'criterion': 'entropy', 'max_depth': 8, 'max_features': 1,
 'splitter': 'best', 'min_samples_split': 2, 'min_samples_leaf': 1,
 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0,
 'class_weight': 'balanced'}
Fitting model in fold 1 of 5. Testing from 1991-03-28 to 1994-01-10.
Score:  -0.7549925441321863
Fitting model in fold 2 of 5. Testing from 1994-07-05 to 1996-10-23.
Score:  -0.7240491144368829
Fitting model in fold 3 of 5. Testing from 1997-04-18 to 1999-08-13.
Score:  -0.727157711743268
Fitting model in fold 4 of 5. Testing from 2000-02-04 to 2002-06-06.
Score:  -0.8191915540550379
Fitting model in fold 5 of 5. Testing from 2002-11-26 to 2005-03-28.
Score:  -0.7828422590967171
Average score is -0.7616 +/- 0.0358.
```

Figure 6.18: An extract of the hyperparameter tuning process of a decision tree with the S&P 500 dataset. The classifier is evaluated with the negative log loss.

```
Testing parameter combination 9 of 18.
{'criterion': 'entropy', 'max_depth': None, 'max_features': None,
 'splitter': 'best', 'min_samples_split': 2, 'min_samples_leaf': 1,
 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0,
 'class_weight': 'balanced'}
Fitting model in fold 1 of 5. Testing from 1991-03-28 to 1994-01-10.
Score:  -17.088612390165203
Fitting model in fold 2 of 5. Testing from 1994-07-05 to 1996-10-23.
Score:  -17.18225351094958
Fitting model in fold 3 of 5. Testing from 1997-04-18 to 1999-08-13.
Score:  -16.934888278206074
Fitting model in fold 4 of 5. Testing from 2000-02-04 to 2002-06-06.
Score:  -16.529632779757367
Fitting model in fold 5 of 5. Testing from 2002-11-26 to 2005-03-28.
Score:  -19.25711391734552
Average score is -17.3985 +/- 0.9557.
```

Figure 6.19: An extract of the hyperparameter tuning process of a decision tree with the S&P 500 dataset. The classifier is evaluated with the negative log loss.

There is no big difference in between the impurity measures. However, the other two hyperparameters influence the performance of the classifier a lot. When set too low, the decision tree is underfitted and performs badly. Setting these two parameters too high

by choosing no restriction ('None') is even worse: the negative log loss is catastrophic as the decision tree is extremely overfitted, see Figure 6.19. Such a decision tree predicts labels with a high predicted probability value. Because there is no restriction, the leaf nodes might only contain a few samples. In that case, the predicted probability of a label for a similar sample is close to 0 or close to 1. If such a prediction is false, the corresponding negative log loss is very small.
Overall, the performance of the decision tree is bad, as it tends to overfit easily.

6.2.2 Random Forest

Next, a random forest is evaluated. To analyse the effect of the labelling horizon, a random forest with fixed hyperparameters and different labelling horizons is tested. Choosing different labelling horizons only has a minor effect on the performance, which can be seen in Table 6.1. In general, it seems to be easier to detect long-term trends, as the performance with the 120-day labelling horizon is slightly better. The goal in Section 6.3 will be to design multiple diversified classifiers. As the performance for all tested labelling horizons is acceptable, all three labelling horizons are used in this thesis.

Labelling Horizon	60	90	120
Accuracy	0.5405	0.5365	0.5397
Precision	0.5367	0.5397	0.5453
F1-Score	0.5398	0.5364	0.5394
Neg. log Loss	-0.6930	-0.6909	-0.6878

Table 6.1: The performance of a random forest in dependence of the labelling horizon.

An extract of the hyperparameter tuning process for a random forest can be seen in Figure 6.20. Compared to the decision tree in Figure 6.18, both the bias and the variance of the model have been improved.

```
Testing parameter combination 11 of 300.
{'n_estimators': 100, 'criterion': 'gini', 'min_weight_fraction_leaf': 0.01,
 'max_features': 1, 'class_weight': 'balanced_subsample',
 'min_impurity_decrease': 0.0, 'n_jobs': -1, 'ccp_alpha': 0.0,
 'max_samples': None, 'bootstrap': True}
Fitting model in fold 1 of 5. Testing from 1991-03-28 to 1994-01-10.
Score:  -0.6934895660238258
Fitting model in fold 2 of 5. Testing from 1994-07-05 to 1996-10-23.
Score:  -0.6754614451650918
Fitting model in fold 3 of 5. Testing from 1997-04-18 to 1999-08-13.
Score:  -0.6893122139408512
Fitting model in fold 4 of 5. Testing from 2000-02-04 to 2002-06-06.
Score:  -0.6971751802985531
Fitting model in fold 5 of 5. Testing from 2002-11-26 to 2005-03-28.
Score:  -0.6949091546556422
Average score is -0.6901 +/- 0.0077.
```

Figure 6.20: An extract of the hyperparameter tuning process of a random forest with the S&P 500 dataset. The classifier is evaluated with the negative log loss.

The effect is even more noticeable when applied on an overfitted classifier. This can be seen when comparing Figure 6.21 to Figure 6.19, as both models have no restriction to the depth of a decision tree or the number of features that can be checked for a split. The random forest is still overfitted, as the performance in fold 5 differs greatly from the other folds. However, the average score has been boosted from -17.4 to -1.3.

```
Testing parameter combination 158 of 300.
{'n_estimators': 50, 'criterion': 'entropy', 'min_weight_fraction_leaf': 0,
 'max_features': None, 'class_weight': 'balanced_subsample',
 'min_impurity_decrease': 0.0, 'n_jobs': -1, 'ccp_alpha': 0.0,
 'max_samples': None, 'bootstrap': True}
Fitting model in fold 1 of 5. Testing from 1991-03-28 to 1994-01-10.
Score:  -0.8144222256813749
Fitting model in fold 2 of 5. Testing from 1994-07-05 to 1996-10-23.
Score:  -0.739637782152737
Fitting model in fold 3 of 5. Testing from 1997-04-18 to 1999-08-13.
Score:  -1.2376554018783814
Fitting model in fold 4 of 5. Testing from 2000-02-04 to 2002-06-06.
Score:  -0.7270546285945348
Fitting model in fold 5 of 5. Testing from 2002-11-26 to 2005-03-28.
Score:  -2.807562642137512
Average score is -1.2653 +/- 0.7936.
```

Figure 6.21: An extract of the hyperparameter tuning process of a random forest with the S&P 500 dataset. The classifier is evaluated with the negative log loss.

To give an overview of how the hyperparameters affect the classifier, the results of the hyperparameter tuning are summarized:

- 'n_estimators': the number of decision trees in the ensemble.
  If the number is set too small, the classifier overfits like a decision tree. Increasing the number of trees improves the model. At a certain number, adding trees has no benefit, as the trees in the ensemble cannot be diversified properly any more. A value around 100 seems to be a good choice.

- 'criterion': the impurity measure.
  There are no major differences in between the Gini impurity and the entropy.

- 'min_weight_fraction_leaf': the minimal weight fraction of samples a leaf is allowed to contain.
  The classifier is overfitted when this value is set too low and underfitted when it is set too high. A value around 0.01-0.05 seems to be a good choice. It has to be set high if 'max_feature' is high and vice versa, as the classifier overfits otherwise.

- 'max_feature': the number of random features a decision tree can evaluate when performing a split.
  The classifier is overfitted when a high value is set. Setting low values works well, as there are many correlated features. A value of 1-3 seems to be a good choice.

- 'class_weight': the sample weight.
  When all samples are weighted equally, the classifier performs badly when trained

on bullish data only. Therefore, the weights are chosen inversely proportional to the class frequencies in the bootstrapped datasets, as described in Section 3.2. This is denoted by 'balanced_subsample'.

The hyperparameter tuning on the STOXX Europe 600 dataset reveals the same optimal parameter combinations. However, the performance on the training dataset is significantly worse. Either European assets behave differently from US-American ones and cannot be described accurately with the underlying set of features, or the time period is disadvantageous. The training dataset spans from 2003 to 2011, which covers the financial crisis from 2007 to 2008 and the European debt crisis afterwards. This might explain why predicting the Edge Ratio Label is more difficult.

6.3 Backtesting

After a classifier has been trained on financial data, it can be used to make predictions for the label of current samples. As the label tries to indicate the future price development of a financial instrument, the classifier can thus help in identifying profitable stocks.

6.3.1 Asset Selection

In Section 5.2, multiple approaches to select assets have been introduced. The first is relying on large market capitalization by investing in all constituents of the index ('No Filter'). The second is to rank assets according to their volatility and to choose a specific basket, here assets with a volatility in between the 20th and the 40th percentile ('Vola Basket'). The third approach is to use the predictions of a machine learning classifier ('ML Selection').
Every 60 trading days, which is roughly equivalent to three months, the portfolio is allowed to be reallocated. This number is chosen arbitrarily to limit transaction costs. For the 'No Filter' strategy, that means that assets that leave the index are replaced with new index members. For the 'Vola Basket' strategy, the volatility is updated and the assets in the new volatility basket are selected. Hence, both strategies do not have to be managed actively. The 'ML Selcetion' strategy, on the other hand, can give daily predictions that could be used to detect trends early. However, to be comparable to the other two strategies, the 'ML Selection' strategy is only allowed to reallocate the portfolio every 60 trading days as well. Lifting this limitation would benefit the 'ML Selection' strategy greatly, as it could better time when to invest and when to go flat. This can be seen in the Appendix D. However, monitoring the financial data more often would be an advantage over the other passively managed trading strategies the 'ML Selection' strategy is compared to.
As mentioned in Subsection 5.2.3, rather than relying on the prediction of a single classifier, the predictions of an ensemble of classifiers are used in order to select profitable stocks with the 'ML Selection' strategy. In this thesis, three different random forests are utilized, which can be seen in Table 6.2.

	Classifier 1	Classifier 2	Classifier 3
Model	Random Forest	Random Forest	Random Forest
Label	Edge Ratio Label	Edge Ratio Label	Edge Ratio Label
Labelling Horizon	60 Days	90 Days	120 Days
Hyperparameter			
n_estimators	75	125	100
criterion	gini	entropy	gini
min_weight_fraction_leaf	0.02	0.03	0.02
max_features	1	2	3
class_weight	balanced_subsample	balanced_subsample	balanced_subsample

Table 6.2: The specifications for the classifiers used to select assets.

After the classifiers have been fitted on the training dataset, they can be used to produce predictions for the probabilities p_1, p_2, and $p_3 \in [0, 1]$ that a current sample has a positive label. The final prediction is the mean of the three predictions:

$$p = \frac{p_1 + p_2 + p_3}{3}.$$

Hence, when $p \geq 0.5$, a positive label would be predicted for the respective sample. In Table 6.3, the merit of an ensemble model can be observed. Combining the predictions of classifiers 1, 2 and 3 can boost the total return while keeping a low average drawdown. Therefore, the ensemble model is used in this thesis when referring to the 'ML Selection' strategy.
The 'Vola Basket' strategy relies on a portfolio size of 20% of the original asset universe. To be comparable, the 'ML Selection' strategy aims for the same desired portfolio size. For example, suppose all assets in the S&P 500 are predicted to have a positive label and thus qualify for an investment. In that case, the 'ML Selection' strategy would only select the 100 assets with the highest value for p. In another scenario, suppose all assets are predicted to have a negative label. In that case, no asset would be selected and the equity would not be invested until the portfolio is reallocated once again. In this thesis, money not invested in the assets is not invested at all, therefore there is no profit from interest.
When the future outlook is bleak, it is possible that the classifiers cannot identify 100 assets to invest in. However, it is possible that individual assets are still predicted to have a positive price development, for example because they are expected to rally more quickly after a financial crisis. In that case, the program is allowed to invest in these assets. Of course, it would not be wise to invest all the equity in only a handful of assets. Therefore, the minimal portfolio size is set to 50. If the classifiers identify 20 assets to invest in, only 40% of the equity can be invested, therefore limiting the risk of losses. Hence, if the classifiers can identify at least 50 assets, the 'ML Selection' strategy is fully invested.

Asset Selection	ML Selection	ML Selection	ML Selection	ML Selection
Classifier	1	2	3	1, 2, 3
Threshold	$p \geq 0.5$	$p \geq 0.5$	$p \geq 0.5$	$p \geq 0.5$
Asset Allocation	Equal Weight	Equal Weight	Equal Weight	Equal Weight
Reallocation	60 Days	60 Days	60 Days	60 Days
Av. Portfolio Size	87.65	88.43	90.09	90.82
Start Equity	1000	1000	1000	1000
End Equity	3362	3948	3795	4194
Total Return (in %)	236.29	294.82	279.54	319.50
Ann. Return (in %)	8.57	9.76	9.46	10.21
Ann. Vola (in %)	19.92	20.94	22.36	20.62
Av. DD (in %)	6.87	6.08	8.20	6.20
Max. DD (in %)	53.58	52.68	60.18	53.01
Sharpe Ratio (Vola)	0.43	0.47	0.42	0.50
Sharpe Ratio (DD)	1.25	1.61	1.15	1.65

Table 6.3: Backtest statistics for chosen strategies, performed on the S&P 500 dataset from 2005-09-14 to 2020-06-19.

The behaviour of the 'ML Selection' strategy can be influenced by setting a threshold for the predicted probability p. In theory, an investment should me made only when the classifiers are reasonably sure that an asset will perform well, which is denoted by a high value for p. Obviously, the range of this value depends on the individual classifier, as an overfitted classifier is more likely to produce very high predicted probabilities. The values tested in this thesis therefore only apply to the classifiers described in Table 6.2 and are influenced by the underlying dataset. The performance of the 'ML Selection' strategy for different thresholds for p in combination with an equal weight ('EW') asset allocation can be seen in Figure 6.22.

For a low threshold, the program is almost always invested. The higher the threshold is set, the longer the portfolio is not invested at all. This can also be seen in Figure 6.23. In general, most of the predicted probabilities for a label seem to be close to 0.5. This can be explained by the design of the classifiers. As described in Section 6.2, the hyperparameters of the classifiers have been chosen such that the negative log loss is maximized. Overly confident false predictions have a very low negative log loss and are therefore avoided, which results in a classifier that is not overfitted.

On the one hand, a high threshold can limit losses. For example, if the threshold is set to 0.55, the portfolio is not invested at all from the end of 2007 to the end of 2008, thus missing a big part of the financial crisis. On the other hand, a high threshold also means that the strategy is overly cautious and misses out on profits, for example from 2010 to 2015 in the case of $p \geq 0.55$. Hence, while it is possible to forecast sudden price shifts like a financial crisis in theory, the model in this thesis is not able to do so precisely. As most of the features are based on lagged trend indicators, this is not surprising.

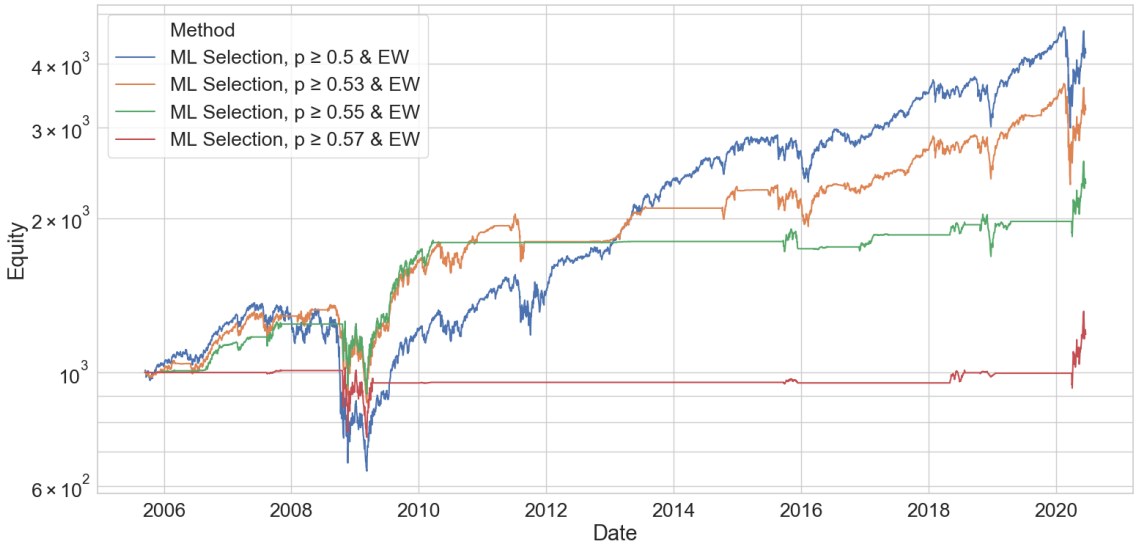

Figure 6.22: Equity curve of multiple 'ML Selection' strategies applied on the S&P 500 dataset with a logarithmic scale.

Figure 6.23: Portfolio size of the 'ML Selection' strategy for different settings on the S&P 500 dataset. The portfolio is reallocated every 60 trading days.

The performance of the different settings can be seen in Table 6.4 with the test statistics that have been defined in Section 5.1. While the lowest threshold $p \geq 0.5$ has the highest return, it also has the highest volatility. The higher the threshold is set, the lower the total return and the lower the volatility and the drawdown becomes. Surprisingly, even though the setting $p \geq 0.55$ produces a strategy that is rarely invested, setting the threshold to 0.5, 0.53 or 0.55 has little effect on the Sharpe ratio based on the volatility. In fact, the Sharpe ratio based on the average drawdown even rises. Only when setting the threshold to 0.57, the Sharpe ratio drops significantly.

Asset Selection	ML Selection	ML Selection	ML Selection	ML Selection
Threshold	$p \geq 0.5$	$p \geq 0.53$	$p \geq 0.55$	$p \geq 0.57$
Asset Allocation	Equal Weight	Equal Weight	Equal Weight	Equal Weight
Reallocation	60 Days	60 Days	60 Days	60 Days
Av. Portfolio Size	90.82	55.17	21.37	6.44
Start Equity	1000	1000	1000	1000
End Equity	4194	3251	2336	1186
Total Return (in %)	319.50	225.15	133.66	18.67
Ann. Return (in %)	10.21	8.32	5.92	1.17
Ann. Vola (in %)	20.62	17.38	12.51	10.71
Av. DD (in %)	6.20	4.23	2.26	4.74
Max. DD (in %)	53.01	36.43	27.87	26.86
Sharpe Ratio (Vola)	0.50	0.48	0.47	0.11
Sharpe Ratio (DD)	1.65	1.97	2.63	0.25

Table 6.4: Backtest statistics for chosen strategies with varying thresholds for p, performed on the S&P 500 dataset from 2005-09-14 to 2020-06-19.

However, the high performance of the strategies can to a large degree be explained by lucky timing, as it is beneficial to reallocate shortly before a price drop occurs. Therefore, the backtest is also influenced by the exact dates the portfolio is reallocated, which is a potential source of overfitting. As the performance of the strategy with a low threshold $p \geq 0.5$ does not rely on going flat at the right time, the strategy probably does not depend on timing. This presumption is analysed in Table 6.5, where the start date of the backtest is shifted by 15, 30 and 45 trading days while keeping the 60 day reallocation rule. While there are differences, the key statistics are roughly similar, which indicates that the strategy is relatively robust.

As the 'ML Selection' strategy with the setting $p \geq 0.5$ has a good performance and is robust to different start dates, this strategy is compared to the other trading strategies. Rather than relying on the classifiers to predict price drops, forcing a full investment seems to be desirable, especially since the risk is controlled with post-processing later in Subsection 6.3.3 anyway.

The backtest for the S&P 500 dataset can be seen in Figure 6.24, a summary of key statistics can be seen in Table 6.6. As a benchmark, the scaled S&P 500 is added. The first thing to notice is that the 'No Filter' strategy that invests in all members of the index with equal weight only has an average portfolio size of 471, in contrast to the 500 mem-

Asset Selection	ML Selection	ML Selection	ML Selection	ML Selection
Threshold	$p \geq 0.5$	$p \geq 0.5$	$p \geq 0.5$	$p \geq 0.5$
Asset Allocation	Equal Weight	Equal Weight	Equal Weight	Equal Weight
Reallocation	60 Days	60 Days	60 Days	60 Days
Start Date	2005-09-14	2005-10-05	2005-10-26	2005-11-16
Av. Portfolio Size	90.82	90.13	91.03	90.44
Start Equity	1000	1000	1000	1000
End Equity	4194	3523	3901	3866
Total Return (in %)	319.50	252.36	290.16	286.60
Ann. Return (in %)	10.21	8.95	9.75	9.72
Ann. Vola (in %)	20.62	20.71	20.13	20.81
Av. DD (in %)	6.20	6.10	5.94	5.94
Max. DD (in %)	53.01	53.08	49.59	49.83
Sharpe Ratio (Vola)	0.50	0.43	0.48	0.47
Sharpe Ratio (DD)	1.65	1.47	1.64	1.64

Table 6.5: Backtest statistics for chosen strategies, performed on the S&P 500 dataset with shifting start dates until 2020-06-19.

bers of the S&P 500 index. As mentioned in the beginning of this chapter, this can be explained by the cleansing of the dataset, because some assets only span a short amount of time and can thus not be processed by the program.

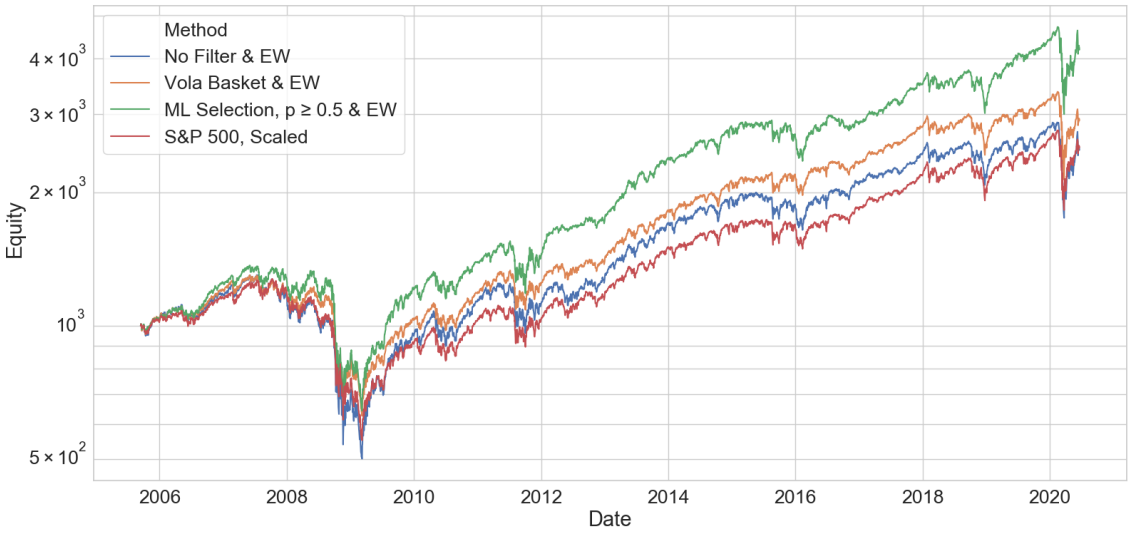

Figure 6.24: Equity curve of selected trading strategies applied on the S&P 500 dataset with a logarithmic scale.

Both the 'Vola Basket' and the 'ML Selection' strategy can outperform the large market capitalization strategy that selects all index members. All strategies are greatly influenced by the financial crisis 2008, which explains the large maximal drawdown. The

Asset Selection	No Filter	Vola Basket	ML Selection	No Filter
Threshold		20th-40th prctl.	$p \geq 0.5$	
Asset Allocation	Equal Weight	Equal Weight	Equal Weight	Index
Reallocation	60 Days	60 Days	60 Days	
Av. Portfolio Size	471.00	94.24	90.82	500
Start Equity	1000	1000	1000	1000
End Equity	2491	2904	4194	2524
Total Return (in %)	149.18	190.48	319.50	152.43
Ann. Return (in %)	6.39	7.50	10.21	6.48
Ann. Vola (in %)	21.92	18.74	20.62	20.21
Av. DD (in %)	9.61	7.20	6.20	9.97
Max. DD (in %)	61.51	51.85	53.01	56.78
Sharpe Ratio (Vola)	0.29	0.40	0.50	0.32
Sharpe Ratio (DD)	0.66	1.04	1.65	0.65

Table 6.6: Backtest statistics for chosen strategies, performed on the S&P 500 dataset from 2005-09-14 to 2020-06-19. As a benchmark, the performance of the scaled S&P 500 is added.

'ML Selection' strategy generates the most profit by far. While the 'Vola Basket' strategy has the least volatility, the 'ML Selection' strategy has the least average drawdown. Overall, the 'ML Selection' strategy has the best performance. This cannot be explained by the classifier's ability to forecast price drops, as the model is fully invested most of the time. Rather, the selection of assets seems to be mostly accurate, especially after a market correction, which can be seen in 2009 and 2019 in Figure 6.24.
Finally, the same classifiers that have been introduced in Table 6.2 are applied on the STOXX Europe 600 dataset. Due to the smaller size of the dataset, the backtest is conducted from 2011 to 2020 only. The corresponding index, here the STOXX Europe 600, is added as a benchmark. What is immediately obvious in Figure 6.25 is that the model is often not invested at all, even for a low threshold of $p \geq 0.5$. This means that the classifiers are reluctant in predicting a positive label. This might be explained by the fact that the training dataset is dominated by the samples following the financial crisis from 2007 onward. The hyperparameter tuning process on the STOXX Europe 600 process also showed a worse performance compared to the S&P 500 dataset in general, so the poor backtest performance was foreseeable. A more close observation of the backtest reveals a bad timing at the beginning, as the model invests as the market prices drop and goes flat when prices start to rally again. Although the 'ML Selection' strategy can almost catch up to the other trading strategies, the final performance is only okay because the model decided to go flat before the prices plummet because of the coronavirus 2020.
As the 'ML Selection' strategy worked best on the S&P 500 dataset when it was almost always invested, a similar approach is applied to the STOXX Europe 600 dataset by lowering the threshold for p even further to zero. In effect, the model cannot go flat any more, as it also invests in assets with a predicted negative label. Rather, it has to rely on its ability to detect the most profitable stocks in a given asset universe. This strategy

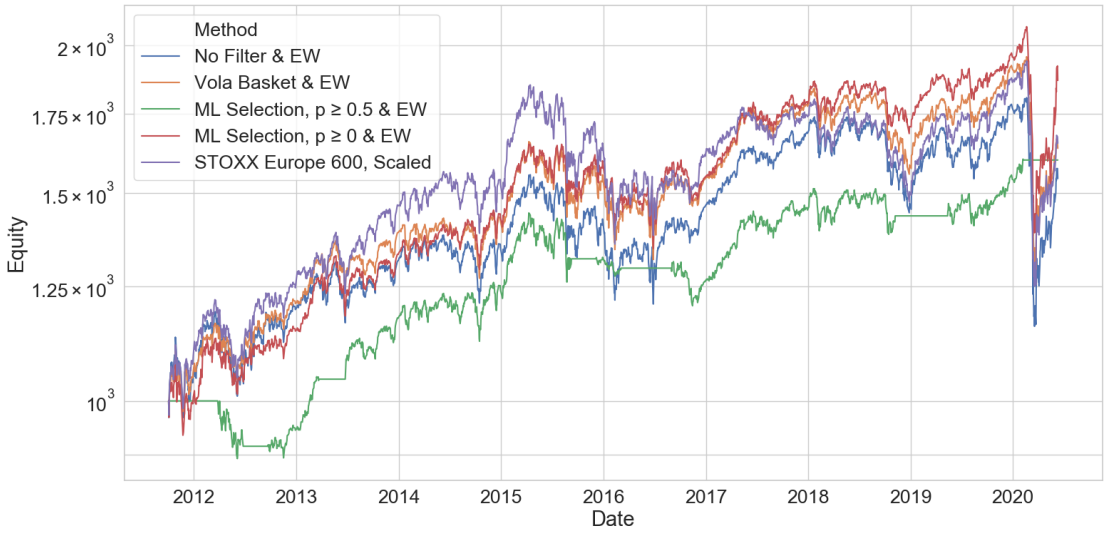

Figure 6.25: Equity curve of selected trading strategies applied on the STOXX Europe 600 dataset with a logarithmic scale.

proves to be somewhat effective, which can also be seen in Figure 6.25. The corresponding test statistics can be seen in Table 6.7. Cutting losses during a market correction is better handled during post-processing, which will be the topic of Subsection 6.3.3. In general, the better the classifiers perform on the training dataset during the hyperparameter tuning, the higher the threshold for p can be set, as the models ability to foresee price drops has probably improved.

Asset Selection	No Filter	Vola Basket	ML Sel.	ML Sel.	No Filter
Threshold		20th-40th prctl.	$p \geq 0.5$	$p \geq 0$	
Asset Allocation	EW	EW	EW	EW	Index
Reallocation	60 Days	60 Days	60 Days	60 Days	
Av. Portfolio Size	564.64	112.83	77.41	112.62	600
Start Equity	1000	1000	1000	1000	1000
End Equity	1542	1636	1599	1867	1652
Total Return (in %)	54.27	63.63	59.94	86.79	65.25
Ann. Return (in %)	5.29	6.03	5.74	7.71	6.15
Ann. Vola (in %)	16.48	14.85	9.40	15.11	17.16
Av. DD (in %)	5.62	4.80	4.09	3.91	7.37
Max. DD (in %)	35.97	33.97	16.56	33.09	35.55
Sharpe Ratio (Vola)	0.32	0.41	0.61	0.51	0.36
Sharpe Ratio (DD)	0.94	1.26	1.40	1.97	0.83

Table 6.7: Backtest statistics for chosen strategies with an equal weight asset allocation, performed on the STOXX Europe 600 dataset from 2011-10-03 to 2020-06-09. As a benchmark, the performance of the scaled STOXX Europe 600 is added.

6.3.2 Asset Allocation

Until now, the equity has been distributed evenly on all assets in the portfolio. As described in Section 5.3, clustering algorithms can be applied to the portfolio to allocate the equity. The method introduced in this thesis is Hierarchical Risk Parity ('HRP'). In theory, correlations of the assets in the portfolio should be taken into account, which should result in a smaller volatility of the equity curve.
The effect of HRP can be seen in Figure 6.26. The difference in between HRP and an equal weight allocation scheme is small. What stands out is the price drop in the end of 2008, where the HRP method manages to limit the loss significantly. During most of the bearish market phases, the HRP method performs better than the equal weight method. During most bullish market phases, it seems to be the other way around. As the beneficial effect in bearish phases is more prominent, the HRP method performs better than the equal weight method.

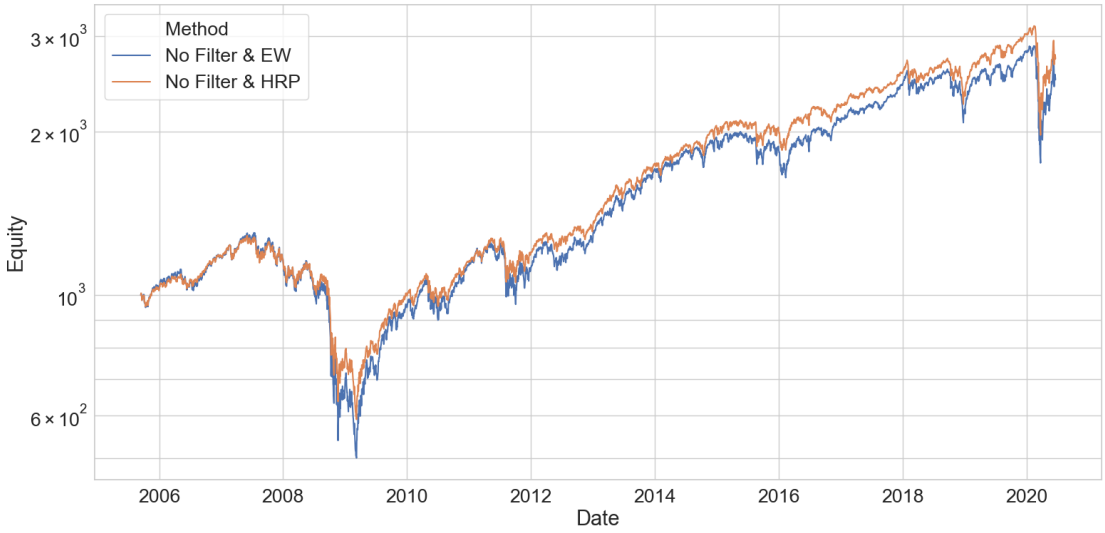

Figure 6.26: Equity curve of selected trading strategies applied on the S&P 500 dataset with a logarithmic scale.

The benefits of the HRP asset allocation can be seen more clearly when looking at the backtest statistics in Table 6.8. A comparison to Table 6.6 shows that the HRP method lowers both the volatility and the drawdown. Only the 'Vola Basket' strategy is unaffected by the HRP allocation. When presented with a diversified portfolio, the HRP algorithm results in an equal weight allocation scheme, which explains the similar results. Nevertheless, when the HRP algorithm adjusts the portfolio weights, it is beneficial, as it lowers the volatility and therefore improves the Sharpe ratio. The results for the STOXX Europe 600 dataset are similar, which can be seen in Table 6.9.

Asset Selection	No Filter	Vola Basket	ML Selection
Threshold		20th-40th prctl.	$p \geq 0.5$
Asset Allocation	HRP	HRP	HRP
Reallocation	60 Days	60 Days	60 Days
Av. Portfolio Size	471.00	94.24	90.82
Start Equity	1000	1000	1000
End Equity	2723	2989	4385
Total Return (in %)	172.33	198.92	338.58
Ann. Return (in %)	7.03	7.71	10.54
Ann. Vola (in %)	18.55	18.18	18.86
Av. DD (in %)	7.69	7.41	4.96
Max. DD (in %)	54.14	52.05	46.63
Sharpe Ratio (Vola)	0.38	0.42	0.56
Sharpe Ratio (DD)	0.91	1.04	2.13

Table 6.8: Backtest statistics for strategies with an HRP asset allocation, performed on the S&P 500 dataset from 2005-09-14 to 2020-06-19.

Asset Selection	No Filter	Vola Basket	ML Selection	ML Selection
Threshold		20th-40th prctl.	$p \geq 0.5$	$p \geq 0$
Asset Allocation	HRP	HRP	HRP	HRP
Reallocation	60 Days	60 Days	60 Days	60 Days
Av. Portfolio Size	564.64	112.83	77.41	112.62
Start Equity	1000	1000	1000	1000
End Equity	1572	1734	1614	1877
Total Return (in %)	57.25	73.47	61.45	87.74
Ann. Return (in %)	5.53	6.76	5.86	7.77
Ann. Vola (in %)	14.17	13.94	8.65	13.38
Av. DD (in %)	4.30	3.81	3.89	3.40
Max. DD (in %)	32.52	31.59	16.91	31.07
Sharpe Ratio (Vola)	0.39	0.49	0.68	0.58
Sharpe Ratio (DD)	1.28	1.78	1.50	2.28

Table 6.9: Backtest statistics for strategies with an HRP asset allocation, performed on the STOXX Europe 600 dataset from 2011-10-03 to 2020-06-09.

6.3.3 Post-Processing

As explained in Section 5.4, post-processing can be applied to an equity curve as a safeguard mechanism. Five trend indicators are applied to the equity curve in this thesis: an envelope around the moving average of the equity, a Bollinger Band, a Donchian channel, the MACD and a trend oscillator. For a more detailed discussion of trend indicators and post-processing, see [Maier-Paape, 2016, p. 1-22].

Trend Indicators

The exact algorithms have been introduced in Subsection 6.1.1 as 'Trend' features. Rather than applying the trend indicator to the price of the index or to the price the individual asset, the trend indicator is applied to the equity curve instead. Again, '1' indicates a positive trend and '0' indicates a negative one.
Unfortunately, at the beginning of the backtest, there are not enough data points of the equity curve to reliably initialize the trend indicators. In contrast to the feature construction, the first values of the trend indicators cannot be discarded, as the trading strategy goes live right away. Waiting until the trend indicators work well means missing out on a lot of potential profit.
A better approach is to decide whether to invest right away based on the current market phase. In order to do that, the trend indicators are applied to the respective index of the dataset, here the S&P 500 or the STOXX Europe 600, as enough data are available at the start of the backtest. For the first 250 trading days, post-processing decisions are thus based on the trend indicators that are applied to the index. Of course, the exact period can vary and depend on the individual trend indicator. Only after this initialization period, the trend indicators are applied directly to the equity curve. This process is explained in more detail for the 'MACDTrend' in the next paragraph.
To vary the trend indicators, different parameters can be set. The 3% envelope around the moving average of the equity, the 2σ Bollinger Band and the Donchian channel are defined by their rolling window. For instance, 'BollingerTrend120(Days)' indicates that a Bollinger band is applied to a 120-day moving average of the equity curve. Similar to the feature construction, the trend is '1' when the upper barrier has been touched last. If the lower barrier has been touched last, the trend is '0'.
Post-processing is applied after the training phase during the backtest. Therefore, time restrictions that have applied to the features do not apply to the trend indicators any more, as all information at the start of a walk-forward backtest can be used. This has implications to the exponential moving average, which does not have to be restricted any more. Therefore, the standard definition for an s-day exponential moving average at day $t \in \{1, \dots, T\}$ can be utilized:

$$\begin{aligned} \mathrm{EMA}_1(p, s) &= p_1 \\ \mathrm{EMA}_t(p, s) &= \frac{2}{s+1} p_t + \left(1 - \frac{2}{s+1}\right) \mathrm{EMA}_{t-1}(p, s), \quad t \geq 2 \end{aligned}$$

where p_t denotes the equity at day $t \in \{1, \dots, T\}$.

MACD Trend

Using this definition of an exponential moving average, the MACD without a time restriction can be defined in dependence of a factor $f \in \mathbb{N}$:

$$\mathrm{MACD}_t(p, f) = \mathrm{EMA}_t(p, 12f) - \mathrm{EMA}_t(p, 26f).$$

Rather than varying the period of days the MACD has access to, the factor f controls whether short- or long-term trends are detected. Similar to the MACD feature, another

exponential moving average is added as a signal line, which also depends on f:

$$\text{signal}_t(p,f) = \text{EMA}_t(p, 9f).$$

The final trend for day $t \in \{251, \dots, T\}$ is calculated similar to the feature construction:

$$(\text{MACDTrend } f \text{ (Factor)})_t$$
$$= \begin{cases} 0 & \text{if } \text{MACD}_t(p,f) \leq \text{signal}_t(p,f) - \sigma^{[250]}_{\text{MACD}(p,f),t} \\ 1 & \text{if } \text{MACD}_t(p,f) \geq \text{signal}_t(p,f) + \sigma^{[250]}_{\text{MACD}(p,f),t} \\ (\text{MACDTrend } f \text{ (Factor)})_{t-1} & \text{else} \end{cases}$$

where $\sigma^{[250]}_{\text{MACD}(p,f),t}$ denotes the 250-day standard deviation of the MACD at day t, which is defined similarly to (6.5). As explained earlier, the trend for day $t \in \{1, \dots, T\}$ is based on the respective index. Suppose the index starts at day $\widetilde{t}_0 \in \mathbb{Z}$ with $\widetilde{t}_0 << 1$. Let $\widetilde{p}_t$ denote the price of the index at day $t \in \{\widetilde{t}_0, \dots, 250\}$. Then, the 'MACDTrend' at day $t \in \{\widetilde{t}_0 + 251, \dots, 250\}$ can be calculated:

$$(\text{MACDTrend } f \text{ (Factor)})_t$$
$$= \begin{cases} 0 & \text{if } \text{MACD}_t(\widetilde{p},f) \leq \text{signal}_t(\widetilde{p},f) - \sigma^{[250]}_{\text{MACD}(\widetilde{p},f),t} \\ 1 & \text{if } \text{MACD}_t(\widetilde{p},f) \geq \text{signal}_t(\widetilde{p},f) + \sigma^{[250]}_{\text{MACD}(\widetilde{p},f),t} \\ (\text{MACDTrend } f \text{ (Factor)})_{t-1} & \text{else} \end{cases}$$

As $\widetilde{t}_0 << 1$, the initialization is arbitrary and can be set to 0:

$$(\text{MACDTrend } f \text{ (Factor)})_{\widetilde{t}_0+250} = 0$$

Trend Oscillator

The final trend indicator, the trend oscillator, can be defined with the standard definition of an exponential moving average as well. The exponential moving average is used to calculate the 'gradient' of the equity p_t at day $t \in \{2, \dots, T\}$

$$\text{grad}_t = \frac{\text{EMA}_t(p, 100) - \text{EMA}_{t-1}(p, 100)}{p_t}$$

and to calculate the oscillator signal

$$\text{osci}_t = \frac{1}{2} + \frac{1}{\pi} \arctan(200 \cdot \text{EMA}_t(\text{grad}, 11)) \in [0, 1].$$

The volatility is calculated similar to Subsection 6.1.1:

$$v_t = \left(\text{osci}_t - \frac{1}{2}\right)^2.$$

However, as there is no time restriction any more, a long-term moving average $\overline{v}^{[1000]}_t$ of the volatility can be used in order to derive an upper and a lower threshold for the oscillator signal. Similar to the other trend indicators, the trend oscillator has an

initialization period of 250 days, so the full 1000-day moving average is not feasible. As a compromise, the moving average is expanded gradually for $t \geq 250$ by calculating $\overline{v}_t^{[\min\{t,1000\}]}$.
The trend oscillator is altered by varying the factor $\alpha \in \mathbb{N}$, which controls the sensitivity of the trend signal. In Subsection 6.1.1, this parameter has been set to 7. For $t \in \{251, \ldots, T\}$, the trend indicator can thus be defined as

$$
\begin{aligned}
&(\text{OscillatorTrend } \alpha \text{ (Alpha)})_t \\
&= \begin{cases} 0 & \text{if osci}_t \leq \frac{1}{2} + \frac{\alpha}{10}\overline{v}_t^{[\min\{t,1000\}]} \\ 1 & \text{if osci}_t \geq \frac{1}{2} + \frac{\alpha}{10}\overline{v}_t^{[\min\{t,1000\}]} \\ (\text{OscillatorTrend } \alpha \text{ (Alpha)})_{t-1} & \text{else} \end{cases}
\end{aligned}
$$

The initialization is similar to the other trend indicators, which has been explained in detail for the 'MACDTrend' indicator in the last paragraph. If $\widetilde{t}_0 \in \mathbb{Z}$ with $\widetilde{t}_0 << 1$ denotes the start of the index price series, the 'OscillatorTrend' for $t \in \{\widetilde{t}_0 + 251, \ldots, 250\}$ is based on the price series of the index and the first value of 'OscillatorTrend' for $t = \widetilde{t}_0 + 250$ is set to 0.

Backtesting

There are three asset selection algorithms that have been introduces in Subsection 6.3.1 ('No Filter', 'Vola Basket', 'ML Selection') and there are two asset allocation methods from Subsection 6.3.2 ('EW', 'HRP'). Hence, there are a total of six different equity curves produced with a backtest for each dataset. In order to test how robust post-processing works, it is applied to all 6 equity curves.
As described in Section 5.4, different settings for the individual trend indicators can be used. In general, these settings adjust how sensitive a trend indicator is. The influence of the number of days the moving average of the 'EnvelopeTrend' indicator is based on can be seen in Figure 6.27. In general, the more days the moving average uses, the later it detects new trends. However, setting the value too low might lead to premature trend signals, which can be seen at the end of 2011.
Rather than over-optimizing post-processing by searching for the perfect setting, a range of effective settings is identified. Trend indicators with different settings are then combined by assigning each trend indicator a share of the equity, as described in Section 5.4. The settings used in this thesis can be seen in Table 6.10.

Trend Indicator	Parameter	Range
EnvelopeTrend	rolling window	100 to 200 by steps of 10
BollingerTrend	rolling window	100 to 200 by steps of 10
DonchianTrend	rolling window	100 to 200 by steps of 10
MACDTrend	factor for (12, 29, 9)	10 to 20 by steps of 1
OscillatorTrend	α (sensitivity)	4 to 14 by steps of 1

Table 6.10: Settings for the trend indicators used for post-processing.

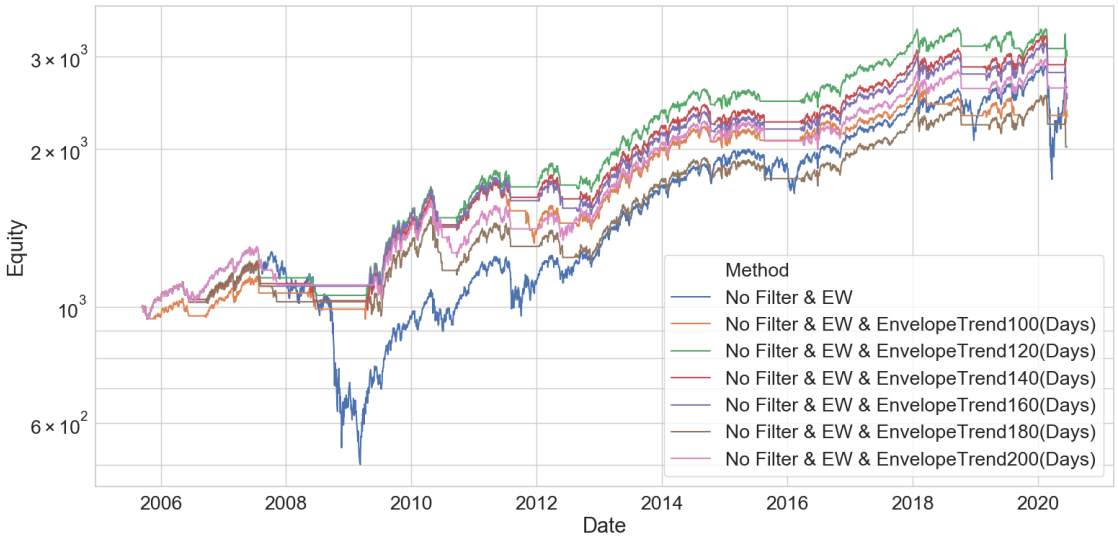

Figure 6.27: Equity curves based on the S&P 500 dataset before and after post-processing with different settings.

Note that all indicators describe a long-term tend. Therefore, the trend only changes rarely. As a single trend indicator is only responsible for a small share of the equity, transactional costs are low and therefore ignored in this thesis.
The influence of this post-processing scheme for the S&P 500 dataset can be seen in Figure 6.28 and Table 6.11.

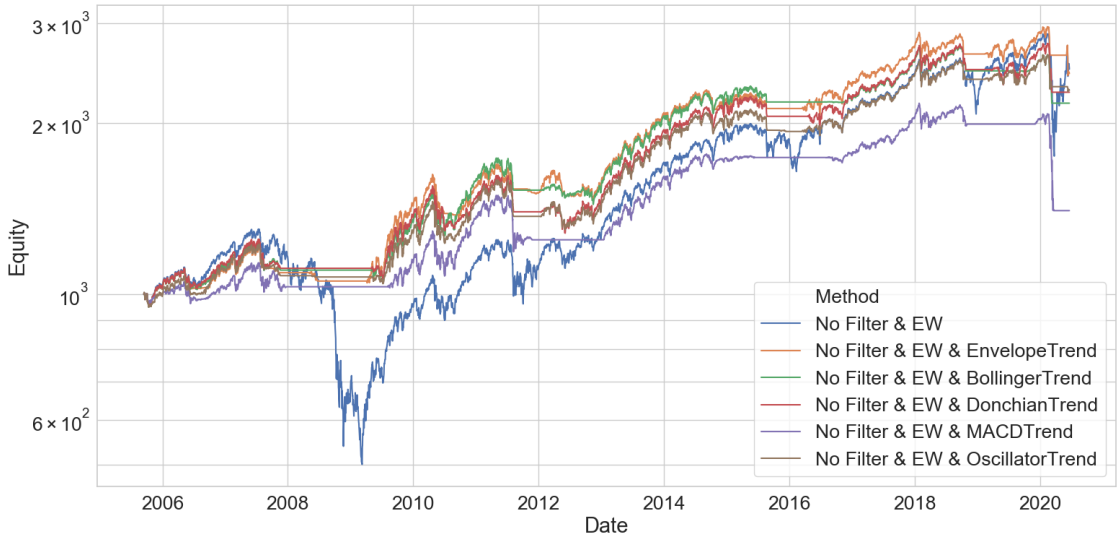

Figure 6.28: Equity curves based on the S&P 500 dataset before and after post-processing.

All trend indicators can limit losses during the financial crisis 2008. Therefore, the equity curve after post-processing outperforms the original 'No Filter & EW' strategy for the most part. However, as the trend indicators rely on lagged data, they are ill equipped for sudden price drops, as seen in 2020. If there is a sudden 'v'-shaped recovery, most

Asset Selection	No Filter	No Filter	No Filter	No Filter	No Filter	No Filter
Asset Allocation	EW	EW	EW	EW	EW	EW
Reallocation	60 Days	60 Days	60 Days	60 Days	60 Days	60 Days
Post-Processing	None	Envelope	Bollinger	Donchian	MACD	Oscillator
Start Equity	1000	1000	1000	1000	1000	1000
End Equity	2492	2448	2170	2269	1403	2287
Total Return (in %)	149.18	144.77	117.01	126.88	40.30	128.65
Ann. Return (in %)	6.39	6.26	5.39	7.71	2.32	5.77
Ann. Vola (in %)	21.92	11.38	10.81	11.47	11.95	10.87
Av. DD (in %)	9.61	5.47	5.93	6.13	6.12	5.88
Max. DD (in %)	61.51	17.97	20.99	20.88	35.33	18.18
Sharpe Ratio (Vola)	0.29	0.55	0.50	0.50	0.19	0.53
Sharpe Ratio (DD)	0.66	1.14	0.91	0.93	0.38	0.98

Table 6.11: Backtest statistics for equity curves with and without post-processing. The backtests have been performed on the S&P 500 dataset from 2005-09-14 to 2020-06-19.

trend indicators stay invested for too long and reinvest too late, which results in relative losses compared to the original strategy. Nevertheless, the overall performance is improved, as drawdowns are reduced significantly.
Post-processing works as a safeguard mechanism that cuts losses when the equity drops too quickly. The better the underlying trading strategy works, the smaller the beneficial effect of post-processing. This can be seen in Figure 6.29 and Table 6.12.

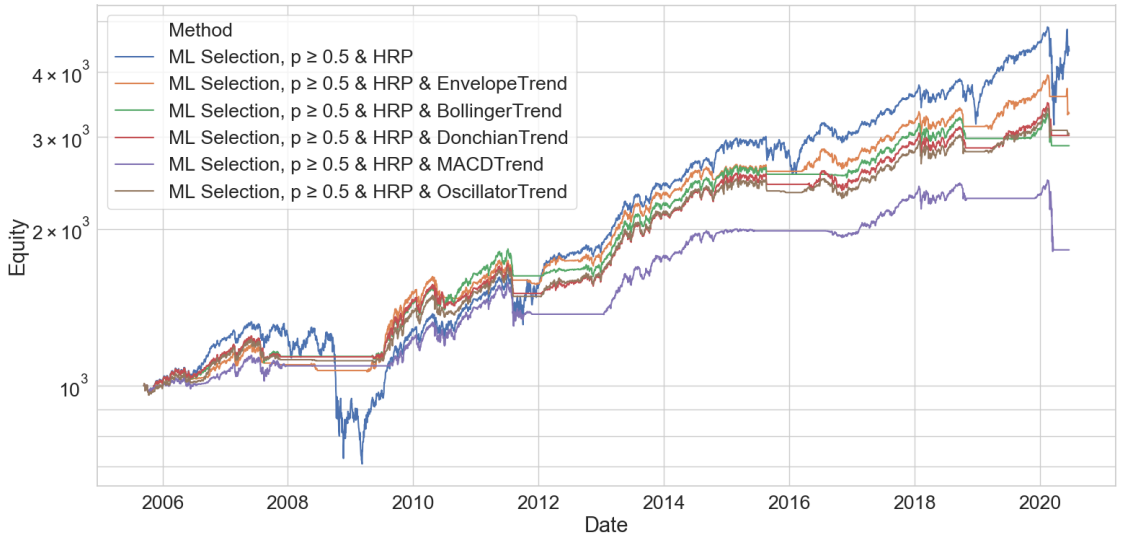

Figure 6.29: Equity curves based on the S&P 500 dataset before and after post-processing.

Although post-processing manages to limit the drawdown during the financial crisis 2008, the original equity curve catches up quickly and manages to outperform the post-

processed equity curves afterwards. As described in Subsection 6.3.1, this can be explained in part by the ability of the 'ML Selection' strategy to identify the most profitable assets after a price correction. If the equity is invested in the assets too late, most of that beneficial effect is lost. However, the overall volatility can still be reduced significantly.

Asset Selection	ML Sel.	ML Sel.	ML Sel.	ML Sel.	ML Sel.	ML Sel.
Threshold	$p \geq 0.5$	$p \geq 0.5$	$p \geq 0.5$	$p \geq 0.5$	$p \geq 0.5$	$p \geq 0.5$
Asset Allocation	HRP	HRP	HRP	HRP	HRP	HRP
Reallocation	60 Days	60 Days	60 Days	60 Days	60 Days	60 Days
Post-Processing	None	Envelope	Bollinger	Donchian	MACD	Oscillator
Start Equity	1000	1000	1000	1000	1000	1000
End Equity	4386	3329	2888	3021	1823	3041
Total Return (in %)	338.58	232.93	188.77	202.13	82.260	204.14
Ann. Return (in %)	10.54	8.50	7.45	7.78	4.15	7.83
Ann. Vola (in %)	18.86	9.27	8.98	9.36	10.26	9.03
Av. DD (in %)	4.96	3.61	4.20	4.26	4.10	3.91
Max. DD (in %)	46.63	16.08	13.16	13.48	27.03	12.18
Sharpe Ratio (Vola)	0.56	0.92	0.83	0.83	0.40	0.87
Sharpe Ratio (DD)	2.13	2.35	1.78	1.83	1.01	2.01

Table 6.12: Backtest statistics for equity curves with and without post-processing. The backtests have been performed on the S&P 500 dataset from 2005-09-14 to 2020-06-19.

Studying the Tables 6.11 and 6.12, the main advantage of post-processing becomes apparent: the drawdown, especially the maximum drawdown, is reduced substantially. In general, the Sharpe ratio can be improved. As seen in Figures 6.28 and 6.29, the 'MACDTrend' indicator detects the last price drop 2020 too late, which might explain the high maximum drawdown and low return. Nevertheless, even the 'MACDTrend' indicator improves both the volatility and the avarage drawdown.
Post-processing works best when there are many bearish phases during the backtest. This can be seen in Figure 6.30, where post-processing is applied on the STOXX Europe 600 dataset. As there are no big equity drops in between 2012 and 2019, the original trading strategy performs best. During the crisis of 2020, most of the post-processed equity curves manage to go flat in time, only the 'MACDTrend' indicator suffers a huge drawdown. Unfortunately, the dataset ends in June, therefore it is unclear how the equity curves develop after the crisis. Up to June 2020, most of the post-processed equity curves have limited the loss while not missing out on a lot of profit.
The missing of the 2008 crisis can explain the under performance of the STOXX Europe 600 dataset compared to the S&P 500 dataset, which can be seen in Table 6.13. Most trend indicators are able to lower the maximum drawdown significantly. However, the average drawdown of the average original equity curve is the lowest. Post-processing with the 'MACDTrend' indicator performs worst by far, as it suffers from the crisis 2020 the most. Furthermore, the annualized return is reduced significantly. Overall, although the volatility can be reduced, post-processing has no significant benefit in a backtest on

the STOXX Europe 600 dataset from 2011 to 2020.

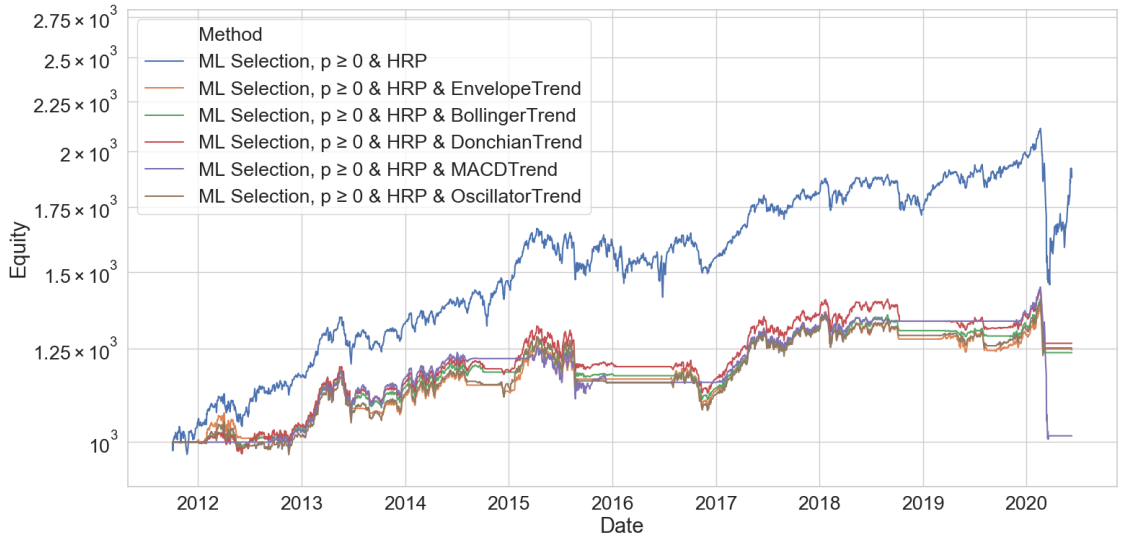

Figure 6.30: Equity curves based on the S&P 500 dataset before and after post-processing.

Asset Selection	ML Sel.	ML Sel.	ML Sel.	ML Sel.	ML Sel.	ML Sel.
Threshold	$p \geq 0$	$p \geq 0$	$p \geq 0$	$p \geq 0$	$p \geq 0$	$p \geq 0$
Asset Allocation	HRP	HRP	HRP	HRP	HRP	HRP
Reallocation	60 Days	60 Days	60 Days	60 Days	60 Days	60 Days
Post-Processing	None	Envelope	Bollinger	Donchian	MACD	Oscillator
Start Equity	1000	1000	1000	1000	1000	1000
End Equity	1877	1245	1238	1265	1015	1251
Total Return (in %)	87.74	24.55	23.75	26.54	1.52	25.15
Ann. Return (in %)	7.77	2.64	2.56	2.84	0.18	2.70
Ann. Vola (in %)	13.38	7.61	7.21	7.65	8.59	7.95
Av. DD (in %)	3.40	5.12	4.19	4.39	3.67	4.99
Max. DD (in %)	31.07	13.71	13.71	14.79	30.38	16.18
Sharpe Ratio (Vola)	0.58	0.35	0.36	0.37	0.02	0.34
Sharpe Ratio (DD)	2.28	0.52	0.61	0.65	0.05	0.54

Table 6.13: Backtest statistics for equity curves with and without post-processing. The backtests have been performed on the STOXX Europe 600 dataset from 2011-10-03 to 2020-06-09.

Conclusion

In this thesis, multiple machine learning algorithms have been analysed and applied to finance. For instance, feature importance has been used to visualize what drives price developments of financial instruments. Furthermore, profitable assets have been detected with classifiers like random forests and a diversified portfolio has been built with clustering algorithms via hierarchical risk parity.
This thesis has shown that it is possible to use machine learning in finance profitably, at least in theory. However, doing so in practice would require a lot of additional work. For example, if the feature importance was used to build and improve trading strategies as described in this thesis, a lot more work would have to go into identifying good features more reliably. Moreover, the feature importance could change over time for different financial instruments, which should thus be investigated as well. Finally, how to combine multiple features to build a coherent trading strategy cannot be answered easily.
Trying to predict stock price movements with random forests has shown that a classifier is only as good as the data it is given. Of course, better algorithms can boost the performance, which could be seen when comparing decision trees to random forests. However, without carefully crafted features that describe a characteristic of a sample, even a random forest cannot be used profitably. Moreover, standard machine learning techniques like cross-validation have to be used carefully, as algorithms on financial data tend to overfit easily. Financial markets are extremely complicated constructs that are highly correlated and change over time, which makes it hard to use machine learning algorithms that have been established in other areas.
In conclusion, machine learning in finance works best when combined with the results of years of financial research. The best trading strategy in this thesis uses a machine learning classifier with established trend indicators as features and post-processing to manage the risk. Thus, machine learning is a valuable tool to learn more about financial markets and to design profitable trading strategies.

Appendix A

Code Snippets

```
def get_osci_envelope(close, window=100, vola_window=1000,
                      osci_window=11, alpha=7):

    close_ema = get_EMA(close, span=window)
    grad = (close_ema - shift_index(close_ema, 1) / close
    grad_ema = get_EMA(grad, span=osci_window)
    osci = 0.5 + arctan(grad_ema * 200) / pi

    vola = (osci - 0.5) ** 2
    vola_ma = get_MA(vola, span=vola_window)

    lower = 0.5 - alpha / 10 * sqrt(vola_ma)
    upper = 0.5 + alpha / 10 * sqrt(vola_ma)

    return osci, lower, upper
```

Figure A.1: Pseudo code for the trend oscillator by Prof. Dr. Stanislaus Maier-Paape.

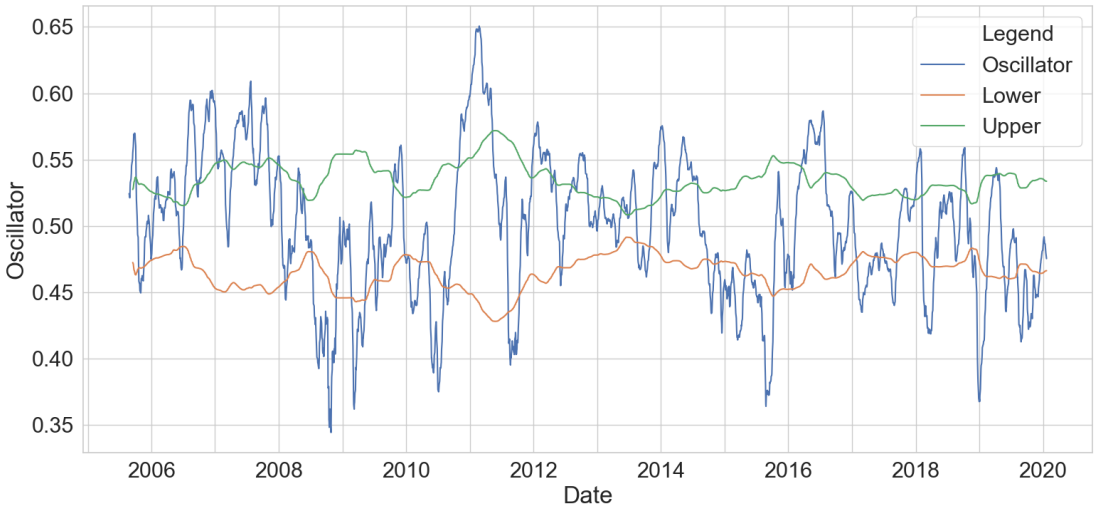

Figure A.2: The 150-day trend oscillator with window=150, vola_window=150, osci_window=11 and alpha=7.

```
import numpy as np


# helper func, perform ewm for rolling in get_EMA
def perform_ema(close, span=10):
    alpha = 2 / (span + 1)
    scaling_factors = np.power(np.repeat(1. - alpha, close.shape[0]),
                               range(close.shape[0])[::-1], dtype=np.float64)
    return np.sum(np.multiply(scaling_factors, close)) / np.sum(scaling_factors)


# get exp weighted moving average with rolling window of limited size
def get_EMA(close, span, lim_window, min_periods=None):
    if lim_window is None:  # no restriction necessary, use standard EMA
        return close.ewm(span=span, adjust=False).mean()
    if min_periods is None:
        min_periods = int(lim_window/4)
    return close.rolling(lim_window, min_periods=min_periods).apply(perform_ema,
                                                                    args=(span,),
                                                                    raw=True)
```

Figure A.3: A fast implementation of an exponential moving average with a limited time window in python.

Appendix B

Feature Plots

Adjusted Trend Indicators

First, the feature importance of the adjusted trend indicators is tested for multiple trend indicators on multiple datasets. This section refers to Subsection 6.1.1, where the other corresponding plots can be seen.

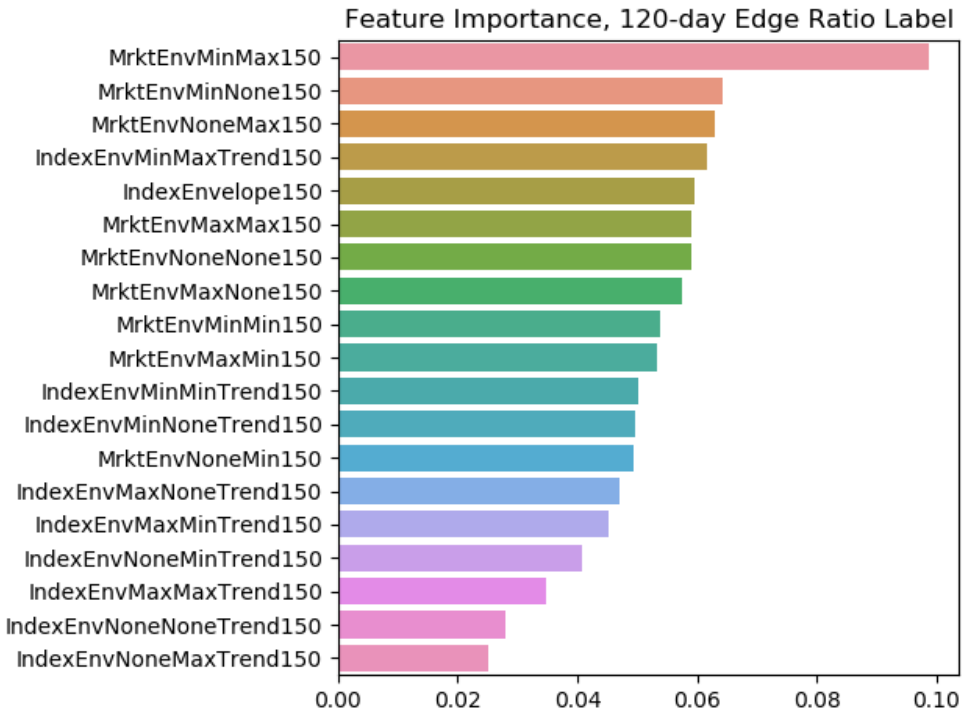

Figure B.1: Feature importance for adjusted envelopes for the S&P 500 dataset.

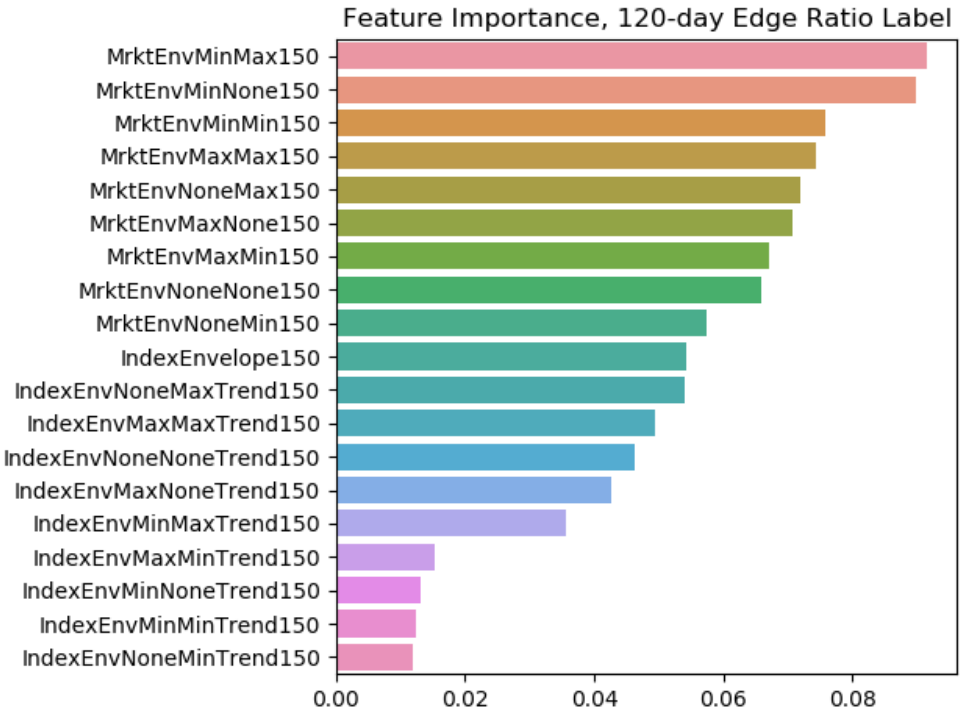

Figure B.2: Feature importance for adjusted envelopes for the STOXX Europe 600 dataset.

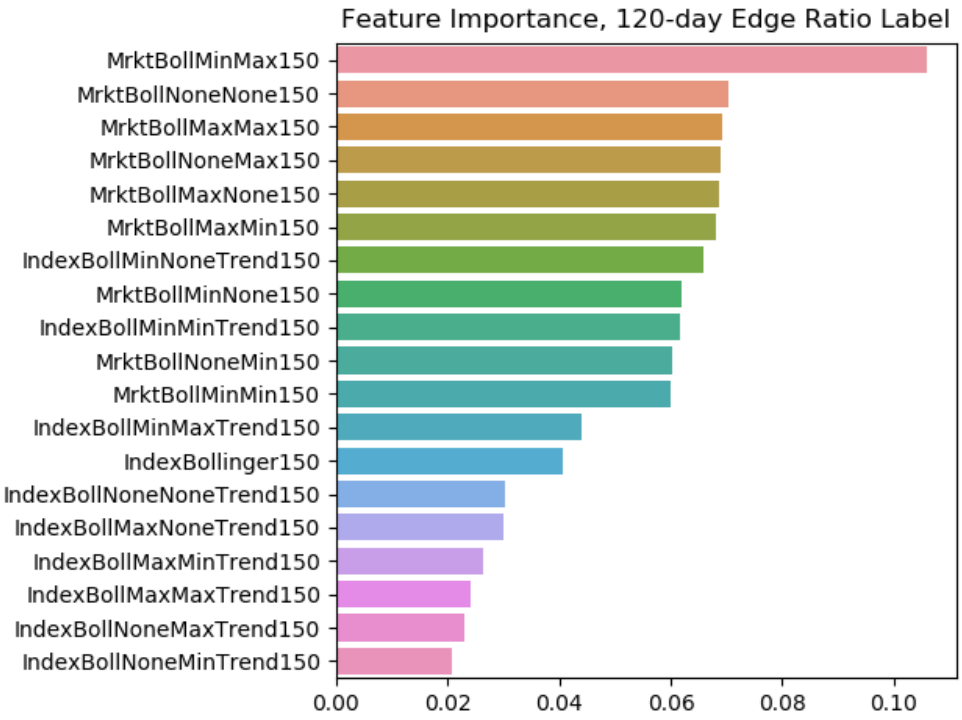

Figure B.3: Feature importance for adjusted Bollinger Bands for the S&P 500 dataset.

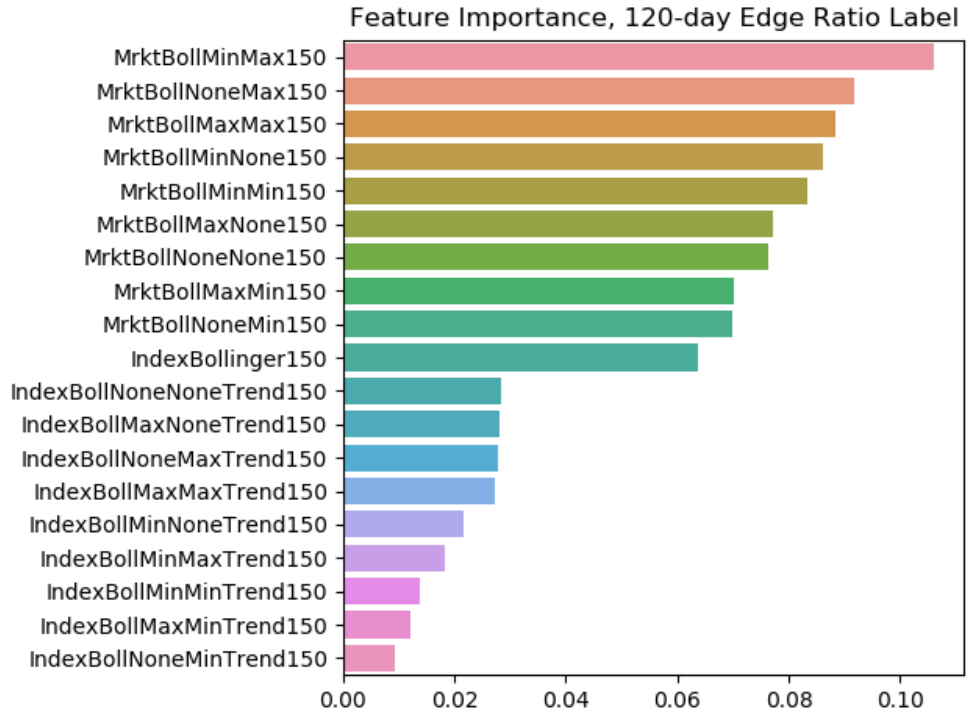

Figure B.4: Feature importance for adjusted Bollinger Bands for the STOXX Europe 600 dataset.

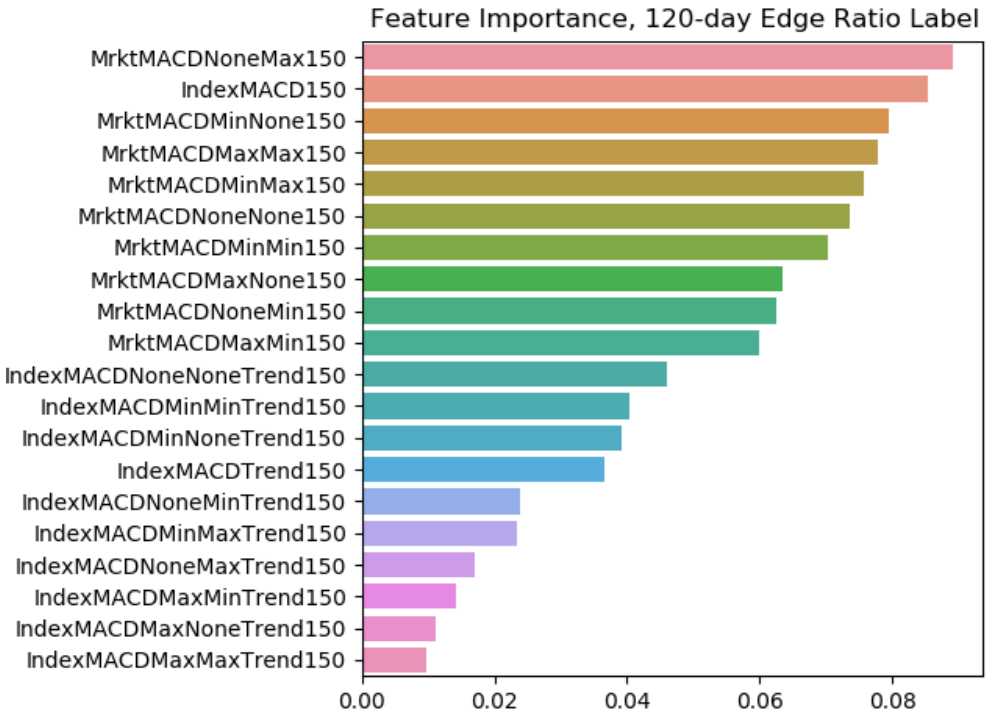

Figure B.5: Feature importance for adjusted MACD features for the STOXX Europe 600 dataset.

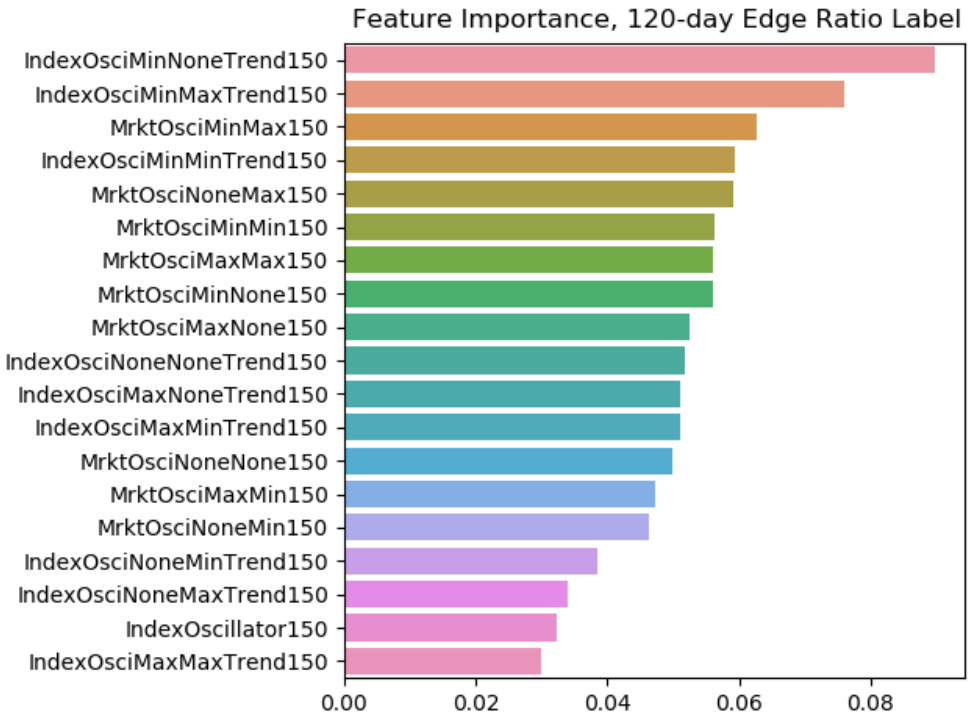

Figure B.6: Feature importance for adjusted trend oscillators for the S&P 500 dataset.

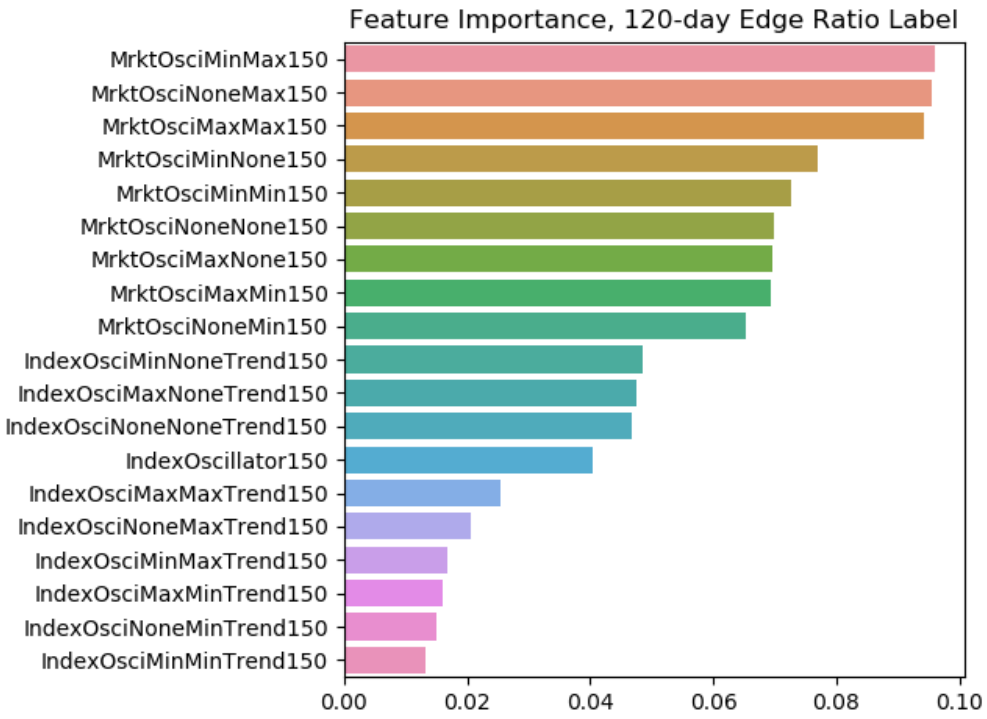

Figure B.7: Feature importance for adjusted trend oscillators for the STOXX Europe 600 dataset.

Feature Importance Overview

An overview of the feature importance of the S&P 500 dataset can be seen in Subsection 6.1.1. Here, the corresponding plots for the STOXX Europe 600 dataset are presented.

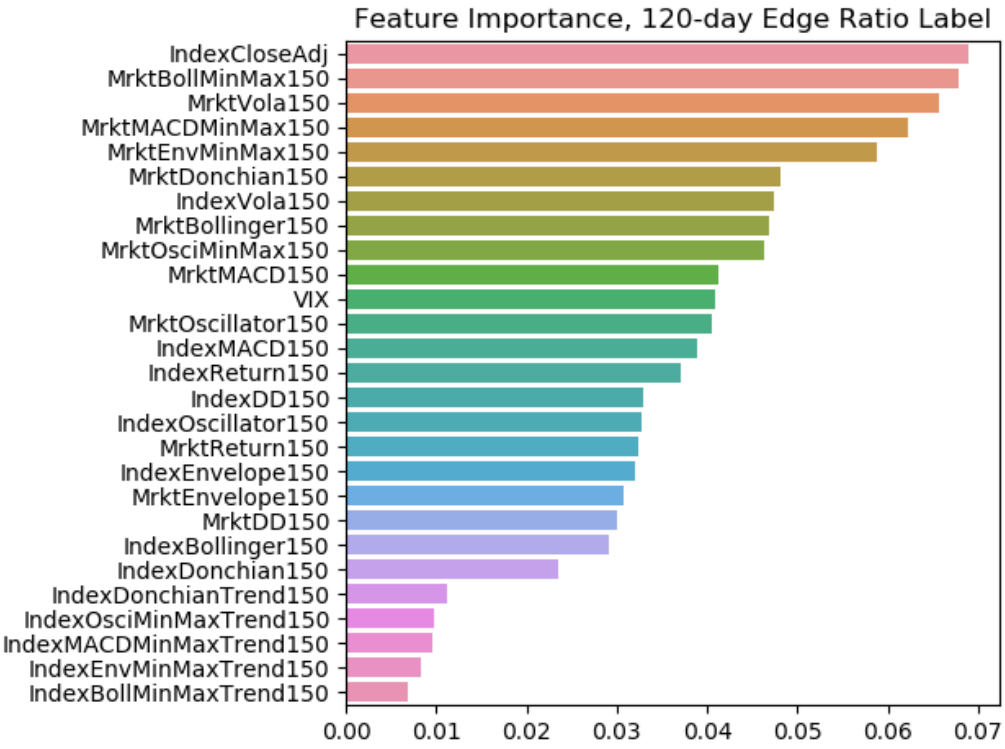

Figure B.8: Feature importance for market features for the STOXX Europe 600 dataset.

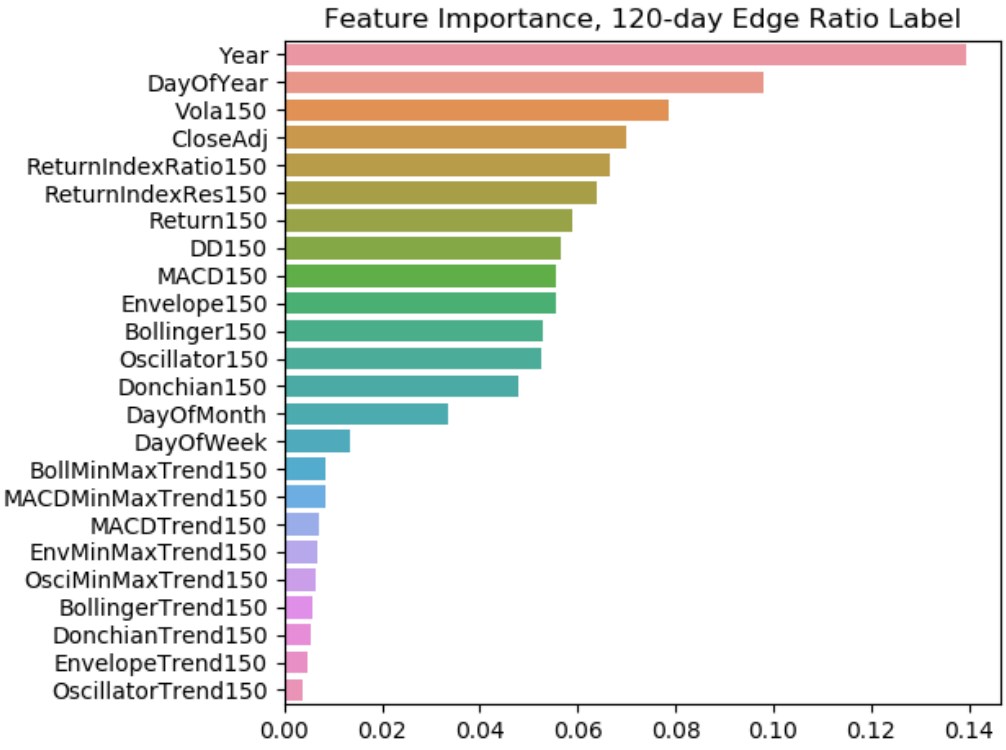

Figure B.9: Feature importance for selected features for the STOXX Europe 600 dataset.

Appendix C

List of Features

The features have been defined in Subsection 6.1.1. For $k \in \{10, 25, 50, 100, 150, 200, 250\}$, the following features are used in the final model:

Time-Based Features

- 'Year'
- 'DayOfYear'

Individual Features

- 'CloseAdj'
- 'Return k' 'ReturnIndexRes k', 'ReturnIndexRatio k'
- 'Vola k', 'DD k'
- 'Envelope k', 'Bollinger k', 'Donchian k', 'MACD k', 'Oscillator k'

Market Features

- 'IndexCloseAdj'
- 'MrktReturn k'
- 'MrktVola k', 'MrktDD k', 'VIX'
- 'MrktEnvMinMax k', 'MrktBollMinMax k', 'MrktMACDMinMax k', 'MrktOsciMinMax k'

Appendix D

Backtest Plots

The influence of different re-allocation periods is visible in the following plots.

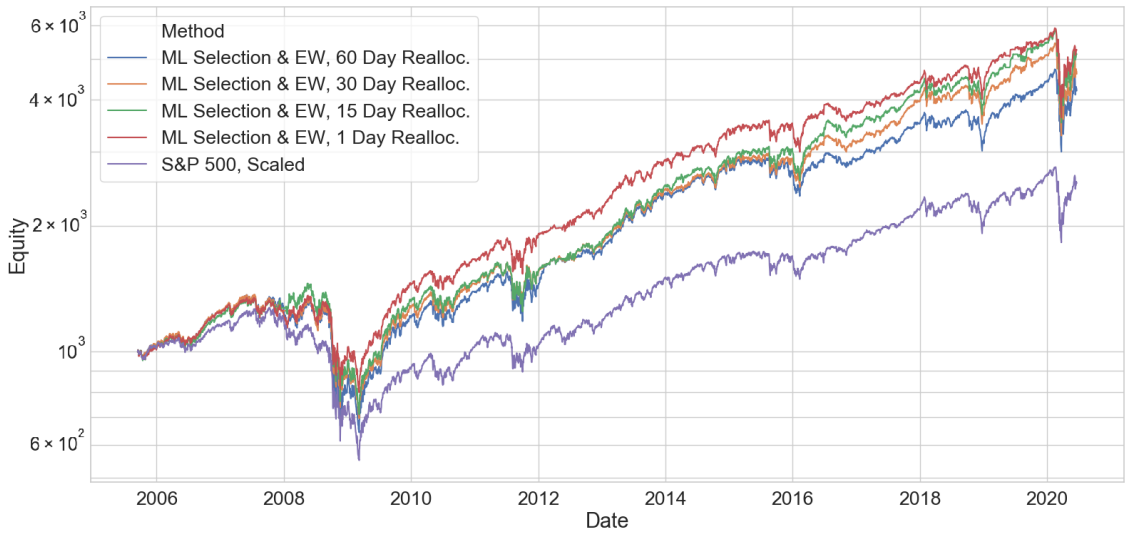

Figure D.1: Equity curve of selected trading strategies with varying reallocation periods, applied on the S&P 500 dataset with a logarithmic scale. The depicted 'ML Selection' strategies have a threshold of $p \geq 0.5$.

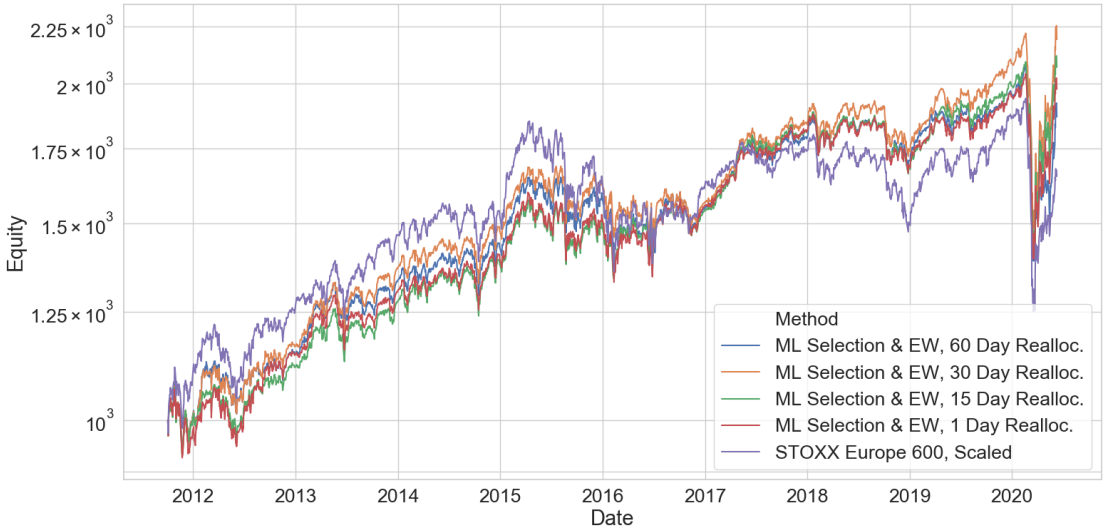

Figure D.2: Equity curve of selected trading strategies with varying reallocation periods, applied on the STOXX Europe 600 dataset with a logarithmic scale. The depicted 'ML Selection' strategies have a threshold of $p \geq 0$.

Bibliography

- Appel, G., 2005. *Technical Analysis: Power Tools for Active Investors*. Financial Times Prentice Hall, Upper Saddle River, NJ, USA.
- Bailey, D. H., Borwein, J. M., López de Prado, M., Zhu, Q. J., 2014. *Pseudo-Mathematics and Financial Charlatanism: The Effects of Backtest Overfitting on Out-of-Sample Performance* Notices of the American Mathematical Society, **61**(5): 458-471
- Bollinger, J., 2002. *Bollinger on Bollinger Bands.* McGraw-Hill, New York, NY, USA.
- Calkin, N., López de Prado, M., 2014. *The Topology of Macro Financial Flows: An Application of Stochastic Flow Diagrams.* Algorithmic Finance, **3**(1): 43–85
- Dahmen, W., Reusken, A., 2006. *Numerik für Ingenieure und Naturwissenschaftler*. Springer, Berlin, Germany.
- De Miguel, V., Garlappi, L., Uppal, R., 2009. *Optimal versus Naive Diversification: How Inefficient is the 1/N Portfolio Strategy*. Review of Financial Studies **22**(5): 1915–1953
- Dixon, M., Klabjan, D., Bang, J. H., 2017. *Classification-based Financial Markets Prediction using Deep Neural Networks*. Algorithmic Finance **6**(3-4): 67-77
- El Sallab, A. Abdou, M., Perot, E., Yogamani, S., 2017. *Deep Reinforcement Learning Framework for Autonomous Driving*. Electronic Imaging **2017**(19): 70-76
- Faith, C. M., 2007. *Way of the Turtle: The Secret Methods that Turned Ordinary People into Legendary Traders*. McGraw-Hill, New York, NY, USA.
- Gebert, T., 2020, April 16. *Kolumne: Der seltsame Erfolg der 16-Wochen Strategie.* Der Aktionär. Retrieved from https://www.deraktionaer.de
- Géron, A., 2019. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*. O'Reilly Media, Inc., Sebastopol, CA, USA.
- Giese, F., 2015. *Nachhaltig erfolgreich traden: Aktienauswahl und Timing - die bewährte Strategie eines Profis*. Börsenmedien AG, Kulmbach, Germany.
- Haase, D., Platen, A., 2019. *Momentum vs. Volatilität: Zwei wichtige Kriterien für die Aktienselektion im Qualitätscheck*. VTAD Award 2019. Retrieved from https://www.vtad.de

- Hamsapriya, T., Renuka, D. K., Chakkaravarthi, M. R., 2011. *Spam Classification based on Supervised Learning using Machine Learning Techniques*. Ictact Journal on Communication Technology, Special Issue on Security and Trust Management in the Digital World **2**(4): 457-462

- Henze, N., 2019. *Stochastik: Eine Einführung mit Grundzügen der Maßtheorie: Inklusive zahlreicher Erklärvideos*. Springer, Berlin, Germany.

- James, G. M., 2003. *Variance and Bias for General Loss Functions*. Machine Learning **51**: 115-135

- James, G. M., Witten, D., Hastie, T., Tibshirani, R., 2013. *An Introduction to Statistical Learning: With Applications in R*. Springer Science+Business Media, New York, NY, USA.

- Kim, K., Ahn, H., 2008. *A recommender system using GA K-means clustering in an online shopping market*. Expert Systems with Applications **34**(2): 1200–1209.

- Kolanovic, M., Krishnamachari, R. T., 2017. *Big Data and AI Strategies: Machine Learning and Alternative Data Approach to Investing*. J. P. Morgan Global Quantitative & Derivatives Strategy, May 2017.

- López de Prado, M., 2018. *Advances in Financial Machine Learning*. John Wiley & Sons, Inc., Hoboken, New Jersey.

- Maier-Paape, S., Zhu, Q., 2018. *A General Framework for Portfolio Theory. Part I: Theory and Examples.* Risks **6**(53): 1-35.

- Maier-Paape, S., Brenner, R., Platen, A., 2016. *Regelbasierter langristiger Trendhandel: Eine statistische Analyse*. 1. Frankfurter Quant-Konferenz, 2016. Retrieved from https://www.instmath.rwth-aachen.de/∼maier/publications_finance/

- March of the Machines, 2019, October 5. *The Economist*, **433**(9163): 18-20.

- Markowitz, H., 1952. *Portfolio Selection*. The Journal of Finance, **7**(1): 77-91.

- Markowitz, H., 1956. *The Optimization of a Quadratic Form Subject to Linear Constraints.* Naval Research Logistics Quarterly **3**: 111-133.

- Mitchell, T. M., 1997. *Machine Learning*. McGraw-Hill, New York, NY, USA.

- Nocedal, J., Wright, S. J., 2006. *Numerical Optimization.* Springer Science+Business Media, New York, NY, USA.

- Oloff, R., 2017. *Wahrscheinlichkeitsrechnung und Maßtheorie*. Springer, Berlin, Germany.

- Papaioannou, I., Lemon, O., 2017. *Combining Chat and Task-Based Multimodal Dialogue for More Engaging HRI: A Scalable Method Using Reinforcement Learning*. ACM/IEEE International Conference on Human-Robot Interaction (HRI), 2017, Vienna, Austria: 365-366

- Quinlan, J. R., 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, USA.
- Raffinot, T., 2018. *Hierachical Clustering-Based Asset Allocation*. The Journal of Portfolio Management Multi-Asset Special Issue 2018 **44**(2): 89-99
- Raileanu, L. E., Stoffel, K., 2004. *Theoretical Comparison between the Gini Index and Information Gain Criteria*. Annals of Mathematics and Artificial Intelligence **41**(1): 77–93
- Raschka, S., Mirjalili, V., 2015. *Python Machine Learning: Machine Learning and Deep Learning with Python, Scikit-Learn and TensorFlow*. Packt Publishing Ltd., Birmingham, Great Britain.
- Van Vliet, P., De Koning, J., 2017. *High Returns from Low Risk: A Remarkable Stock Market Paradox*. John Wiley & Sons Ltd, Chichester, Great Britain.
- Yadav, A., Jha, C. K., Sharan, A., Vaish, V., 2019. *Sentiment Analysis of Financial News Using Unsupervised And Supervised Approach*. Pattern Recognition and Machine Intelligence: 8th International Conference, PReMI 2019, Tezpur, India: 311-319