Ludwig-Maximilians-Universität München

Institut für Statistik

Master's Thesis

Dynamic Investment Strategies with Machine Learning Methods

Thomas Berger

Munich, 05 June 2019

Supervision: Prof. Stefan Mittnik, PhD and Christoph Berninger, M.Sc.

Abstract

One of the main problems in financial machine learning are based on the low signal to noise ratio. In other applications well performing algorithms mistake noise for signal which leads to poor results. This thesis tries to deal with these problems by applying CUSUM filter and labelling observations with triple-barrier method. Another key characteristic of financial time series is their dependency structure. Sample weights are introduced to deal with this issue. Thus, observations are weighted regarding their return attribution and dependency structure. For modelling financial patterns the ensemble methods bagging and random forest are considered in this thesis as well as logistic regression.
In the empirical analysis the DAX between April 1993 and December 2018 is investigated. 14 different features are used to examine the DAX pattern structure. To assess the influence of the features, research is done by measuring feature importance with and without sample weights. Overall, the momentum variables are ranked comparatively high for predicting DAX movements. For backtesting, the combinatorial purged cross-validation is considered as scenario based methodology. For this application, four different models are used with and without sample weights: Logistic regression, bagged logistic regression, bagged trees and random forest. On this occasion, random forest and bagged trees with sample weights yield the best results with respect to the classification measures ROC-AUC and F1. Furthermore, a superior sharpe ratio and maximal drawdown are observed.

Keywords: CUSUM filter, triple-barrier method, stock price predicting, sample weights, bagging, random forest, logistic regression, combinatorial purged cross-validation.

Contents

List of Figures

List of Tables

Notation

a	Parameter in the binomial coefficient
$\mathcal{B}$	Backshift operator
B	Number of bootstrap samples
b	Bet size
C	Regularization parameter in logistic regression
c_t	Number of concurrent labels at time t
d	Real value fractional differentiation parameter
e	Embargo time
$f()$	Deterministic function for generating outputs
$G()$	CDF of standard gumbel distribution
h	Threshold parameter for CUSUM filter
$\mathcal{I}(.)$	Impurity function
$J()$	Penalty function in logistic regression
K	Number of folds in cross-validation
k	Number of test datasets in the CPCV framework
$L(.)$	Arbitrary loss function
l	Vertical barrier time span
M	Number of rectangles
N	Number of partitions in the CPCV framework
$\hat{O}_m(x)$	Predicted class of a tree
P_t	Close price at time t
$\hat{p}_{mq}(m)$	Relative freq of category q in the response node m
$\tilde{p}$	Largest probability provided by a model for several classes
Q	Number of categories in a CART
$\mathcal{R}$	Half-space
R_m	Node m in a tree

R_t^{net}	Net return at time t
r_t	Log-return at time t
r_f	Risk free rate
S_t	Cumulative sum in CUSUM filter
$\mathcal{T}$	Threshold in FFD computation
T_m	Number of observations in node m
$u_{t,i}$	Uniqueness of a label i at time t
w	Sample weight parameter
$X_{T,p}$	Feature matrix
y_t	Potential label for observation t
Z	Arbitrary time series

α	Normalizing constant
β	Normalizing constant
$\chi_R(x)$	Indicator function
Δ	First difference operator
δ	Parameter for lagged observations in autoregressive model
ϵ	White Noise
η	Linear predictor
γ	Euler-Mascheroni constant
ι	Portion of data in the CPCV framework
κ	Threshold parameter in fixed-time horizon method
Λ	ADF test statistic
λ	Decay parameter in EWMA model
μ	Mean
ω	Weight parameter for fractional differentiation
Π	Parameter in ADF model
π	Conditional probability
Φ	Standard normal distribution

ϕ	Sigmoid function
Ψ_i	Wealth at time i
ϱ	Autocovariance function
ρ	Correlation parameter
σ^2	Variance parameter
τ	Index for threshold overrun
θ	Coefficients in logistic regression
φ	Number of targeted paths
ξ	Number of non-overlapping outcomes
ζ	Bet size test statistic

Running Indices

$i = 1, \ldots, B$	Number of bootstrap samples
$i = 1, \ldots, I$	Number of filtered observations
$j = 1, \ldots, p$	Number of features in X
$i = 1, \ldots, k$	Number of test sets in the CPCV framework
$m = 1, \ldots, M$	Number of nodes in a decision tree
$s = 1, \ldots, \mathcal{S}$	Number of trials
$t = 1, \ldots, T$	Number of observations

Abbreviations

ADF	**A**ugmented **D**ickey-**F**uller
CART	**C**lassification **A**nd **R**egression **T**ree
CDF	**C**umulative **D**istribution **F**unction
CPCV	**C**ombinatorial **P**urged **C**ross-**V**alidation
CUSUM	**Cu**mulative **Sum**
CV	**C**ross-**V**alidation
EWMA	**E**xponential **W**eighted **M**oving **A**verage
FFD	**F**ixed-Width Window **F**rac**D**iff
IID	**I**ndependent and **I**dentically **D**istributed
MDA	**M**ean **D**ecrease **A**ccuracy
MDI	**M**ean **D**ecrease **I**mpurity
SFI	**S**ingle **F**eature **I**mportance
PCA	**P**rincipal **C**omponent **A**nalysis
SR	**S**harpe **R**atio
MaxDD	**Max**imum **D**raw**d**own
TP	**T**rue **P**ositive
TN	**T**rue **N**egative
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
WF	**W**alk **F**orward

1 Introduction

Machine Learning methods have improved modelling in various aspects by anticipating non-linear effects in data. Whereas machine learning algorithms succeeded in many applications, predicting stock movements is quite challenging. Although these sophisticated models have been used in hedge funds for decades, its records in quantitative finance are mixed. As stated by Asmundsson (2018), the Eurekahedge AI Hedge Fund Index tracks the returns of 13 hedge funds which considers machine learning in their models. The Eurakhedge benchmark was not able to beat the market in the last five years since this index has gained 7% per year, whereas the S&P 500 returned 13% annually. Therefore, investing in financial markets by using machine learning methods does not guarantee outstanding success.

Applying machine learning methods to financial data has to be done under the consideration of the specific differences between stock return data and other datasets. Financial time series exhibits special characteristics, like heteroskedasticity, low signal-to-noise ratio and non-normality of the returns. A standard approach would be to apply an ARMA model to an integer differentiated time series to deal with the possible occurrence of serial correlation. However, the performance of this model is far away of satisfying. Models which can deal with non-linear data might improve modelling the behaviour of financial time series. But as explained above, although machine learning models can lead to impressive results, this does not necessary hold for financial time series.

As described by López De Prado (2018), low signal-to-noise ratio leads to false discoveries if machine learning algorithms are misused. Applying a sophisticated machine learning model like “Xgboost” or some deep learning method to a log-return time series of an arbitrary stock will almost surely fail because the algorithm will confuse statistical fluke with patterns. Therefore, the special characteristics of financial data have to be kept in mind. For instance, filtering financial data can enhance necessary signals to apply machine learning to stock data more successfully.

This work investigates the potential benefit of data preparing methods as stated by López De Prado (2018). In contrast to applying machine learning methods to high frequency data, this thesis examines different data treatment, modelling and backtesting methods on daily index data. Investigations in this field can lead to a deeper understanding of market movements and as a result to smarter investment decisions.

2 Data Preparation and Preprocessing

2.1 Stationarity and Fractional Differentiation

One important step in working with time series is to make them stationary. A time series is defined as *strictly stationary* if for every V and T the distribution of $\{Z_1, \ldots, Z_T\}$ and $\{Z_{1+V}, \ldots, Z_{T+V}\}$ are the same. This means that the probability distribution of a sequence of T observations does not depend on their time origin. Hence, all aspects of its behaviour are unchanged by shifts in time. This is a rather strong assumption and in most cases *weak stationarity* is satisfactory for time series modelling. A time series $\{Z_1, \ldots, Z_n\}$ is weakly stationary if the conditions

- $\mathbb{E}[Z_t] = \mu$ for all t,
- $\mathrm{Var}[Z_t] = \sigma^2$ for all t and
- $\mathrm{Cov}(Z_t, Z_s) = \varrho(|t - v|)$ for all t and v and some function $\varrho()$

hold, whereas $\varrho()$ represents the autocovariance function. Thus, the mean and the positive finite variance does not depend on t. Moreover, the covariance depends on the *lag* between two time points, but not on the timepoint t itself. This is called weak stationary since the conditions concern only mean, variance and covariance and not other distributional characteristics such as skewness and kurtosis. One advantage of stationary time series is that it can be modelled with relatively few parameters. For instance, the mean can be modelled by taking the average $\overline{Z}$ because it does not change in time, whereas in non-stationary cases for each timepoint another mean has to be estimated (Ruppert & Matteson 2015).

Hence, stationarity can help to find a parsimonious model. For mapping an unknown observation to a collection of known labelled observations, supervised algorithms require stationary features to infer a label from observed examples. Therefore, in financial statistics it is a popular approach to make a price series stationary by computing its returns or log-returns. However, as stated by López De Prado (2018), returns have no *memory* and are not correlated with the original price series. In contrast to returns, price series have memory because every value is dependent upon a long history of previous levels. To reach stationarity for further statistical analysis, a popular approach is to use integer differentiation which cuts off the memory of the series. Hence, important patterns get lost to get stationarity. One method to preserve memory and get stationarity is called *fractional differentiating.* Consider the binomial series for a real number d, $(1+z)^d = \sum_{a=0}^{\infty} \binom{d}{a} z^a$. By incorporating the backshift operator $\mathcal{B}$ into the binomial series the fractional model

$$\begin{aligned}(1-\mathcal{B})^d &= \sum_{a=0}^{\infty} \binom{d}{a} (-\mathcal{B})^a = \sum_{a=0}^{\infty} \frac{\prod_{i=0}^{a-1}(d-i)}{a!}(-\mathcal{B})^a \\ &= \sum_{a=0}^{\infty} (-\mathcal{B})^a \prod_{i=0}^{a-1} \frac{d-i}{a-i} \\ &= 1 - d\mathcal{B} + \frac{d(d-1)}{2!}\mathcal{B}^2 - \frac{d(d-1)(d-2)}{3!}\mathcal{B}^3 + \dots\end{aligned}$$

is derived. As stated by Hosking (1981), fractional differentiation is a tool to express the long memory in a time series data by weighting previous values with a certain factor. Considering parameterisation of the weights with

$$\omega = \{1, -d, \frac{d(d-1)}{2!}, \frac{d(d-1)(d-2)}{3!}, \dots, (-1)^a \frac{\prod_{i=0}^{a-1}(d-i)}{a!}\}$$

if they are applied to the real values $Z = Z_t, Z_{t-1}, Z_{t-2}, Z_{t-3}, \dots, Z_{t-a}, \dots$, long memory can then be expressed through

$$\tilde{Z}_t = \sum_{a=0}^{\infty} = \omega_a Z_{t-a}.$$

Considering parameter d, the weights ω are crucial for understanding the concept of long memory. As mentioned before, d controls the amount of differentiation within the fractional differentiation method. An integer differentiation leads to $\prod_{i=0}^{a-1} \frac{d-i}{a!} = 0, \forall a > d$, hence memory is cancelled beyond a point greater than d (López De Prado 2018).

For example, considering $d = 1$, a common approach to derive log-price returns, results in $\omega = 1, -1, 0, \dots, 0$. Figure 2.1 shows the weights for different d inputs.

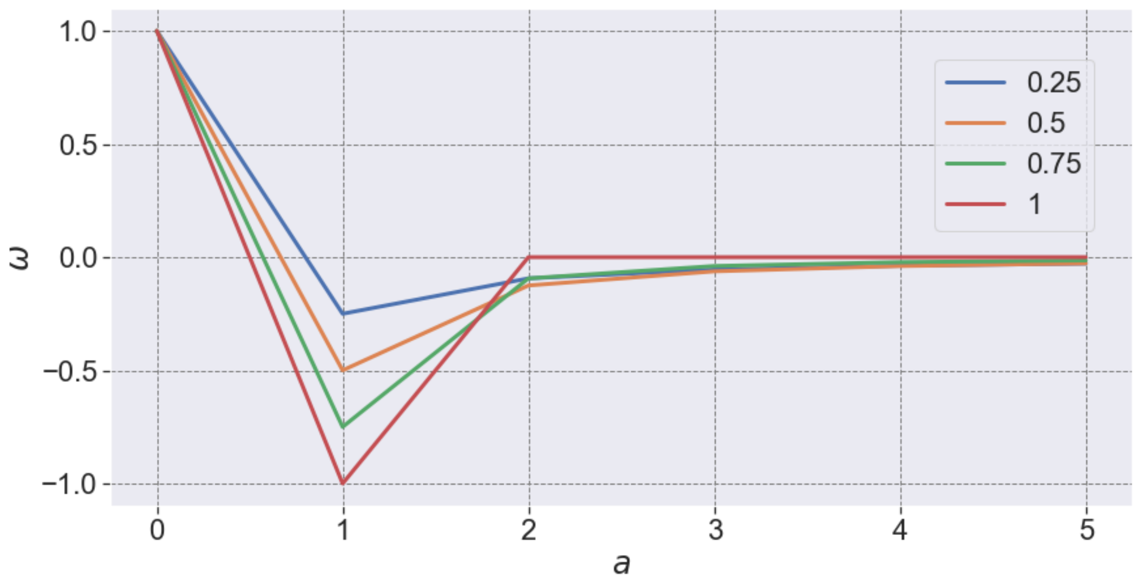


Figure 2.1: Weights evolution as a function of d (López De Prado 2018, p.78)

The first weight ω_0 equals to one by definition. The other weights can be generated iteratively as

$$\omega_a = -\omega_{a-1}\frac{d-a+1}{a}.$$

Observing $|\frac{\omega_a}{\omega_{a-1}}| = |\frac{d-a+1}{d}| < 1$, the weights converge asymptotically towards zero, if $a > d$ and $\omega_{a-1} \neq 0$. For a positive d and $a < d+1$, $\frac{d-a+1}{a} \geq 0$ holds and makes the initial weights alternate in sign. An even Int$[d]$ results in $\lim\limits_{a\to\infty} \omega_a = 0^-$. On the other side $\lim\limits_{a\to\infty} \omega_a = 0^+$ if Int$[d]$ is odd. Hence, memory disappears with increasing a. However, the alternation of weight signs is necessary to make $\{\tilde{Z}_t\}_{t=1,\dots,T}$ stationary (López De Prado 2018).

In practice there is a limited number of observations $\{Z_t\}_{t=1,\dots,T}$ so that fractional differentiation cannot be computed on an infinite series of weights. A method to estimate the weights is called *Fixed-Width Window Fracdiff* (FFD). The main idea is to drop weights after a weight falls below a pre-defined absolute threshold $\mathcal{T}$. This is equivalent for finding the first $l^\star$ such that $|\omega_{l^\star}| \geq \mathcal{T}$ and $|\omega_{l^\star+1}| \leq \mathcal{T}$, setting a new variable $\tilde{\omega}_a$

$$\tilde{\omega}_a = \begin{cases} \omega_a & \text{if } \; a \leq l^\star \\ 0 & \text{if } \; a > l^\star \end{cases}$$

and $\tilde{Z}_t = \sum_{a=0}^{l^\star} = \tilde{\omega}_a Z_{t-a}$ for $t = T - l^\star + 1, \ldots, T$.

The advantage of this method is to consider the same vector across all estimates of $\{\tilde{Z}_t\}_{t=l^\star,\ldots,T}$. Hence, the negative drift which is caused by the convergence behaviour is avoided becaue no $\{\tilde{Z}_t\}$ is computed by a greater window of values. Performing the FFD method with a real value parameter d usually leads to a reduced differentiated dataset because every value is for example computed by its 90 lagged values. For this reason, the first 90 values can not be used for the later analysis.

As mentioned before, there is a trade-off between memory and stationarity. The minimum parameter $d^\star$ is used to achieve stationarity and also to preserve as much memory as possible. A natural procedure would be to compute an Augmented Dickey-Fuller statistic on a pre-defined amount of parameters d and consider the value $d^\star$ which crosses the 95% confidence value of an *Augmented Dickey-Fuller Test* (ADF test).

As described by Ruppert & Matteson (2015), the ADF test is an extension of the Dickey-Fuller test, which concludes whether a time series processes an unit root and is consequently not stationary. The Augmented Dickey-Fuller test is based on an autoregressive model, at which the number of lags can be conducted by Akaike information criterion.

However, as suggested by López De Prado (2018), in the empirical analysis the formula

$$\Delta Z_t = \Pi_0 + \Pi_1 Z_{t-1} + \delta \Delta Z_{t-1} + \epsilon_t$$

is sufficient to test for stationarity. Δ represents a first difference operator such $Z_t = Z_t - Z_{t-1}$. As described by Alexander (2001), the test statistic results from

$$\Lambda = \frac{\hat{\Pi}}{\sqrt{\text{Var}[\hat{\Pi}]}}$$

where the hypothesis are defined by $H_0 : \Pi = 1$ and $H_1 : \Pi < 1$. Based on MacKinnon

(2010), the $0.01, 0.05$ and 0.1 quantiles of the limit distribution Λ are $-3.43, -2.86$ and -2.57, respectively.

2.2 CUSUM Filter

The following section covers a filtering method for financial data. Therefore, the main characteristics of the CUSUM filter methodology are explained. Furthermore, the filter is applied to a DAX time series.

Sampling relevant observations for features is one important task before applying machine learning methods to financial data. To achieve this objective, event-based sampling is performed. A not negligible number of traders at the financial markets are chartist who base their trading strategies on price charts. When chart trader notice an upward (downward) trend in financial market, they evaluate it as a sign to take a long (short) position in order to benefit from a trend. One concept to filter relevant examples is the CUSUM filter. Introduced by Page (1954) the filter was usually considered for quality-control methods, though it can also be used for financial time series. In general, it measures if a mean value shifts away from a target value. Hence, people working in quality-control gets an early alarm when quality shifts down or upwards. In financial markets CUSUM filter is based on the rationale that it is more likely for a stock to move up (down) further if the stock previously moved up (down) (Lam & Yam 1997).

As mentioned before, the CUSUM procedure is designed to detect a shift in the mean value of a measured quantitiy away from a target value. More precisely, the filter always samples an observation if a pre-defined threshold is activated. This is defined by

$$S_t \geq h \Leftrightarrow \exists \tau \in [1, t] \left| \sum_{i=\tau}^{t} (Z_i - \mathbb{E}_{i-1}[Z_t]) \geq h. \right.$$

The threshold is determined through the parameter h. Hence, the symmetric CUSUM filter can be defined by cumulative sums of IID observations $\{Z_t\}_{t=1,\dots,T}$

$$\begin{aligned} S_t^+ &= \max\{0, S_{t-1}^+ + Z_t - \mathbb{E}_{t-1}[Z_t]\}, \; S_0^+ = 0 \\ S_t^- &= \min\{0, S_{t-1}^- + Z_t - \mathbb{E}_{t-1}[Z_t]\}, \; S_0^- = 0 \\ S_t &= \max\{S_t^+, -S_t^-\}. \end{aligned}$$

According to the cumulative sums, the filter recognizes upward and downward trends in the observed series. Hence, bars are only sampled if and only if $S_t \geq h$ and after sampling a bar, S_t is reset. For all CUSUM filter applications in this thesis $\mathbb{E}_{t-1}[Z_t] = Z_{t-1}$ is assumed (López De Prado 2018).

Figure 2.2 illustrates the filtering of observations with a DAX log-return time series between January 2002 and January 2003. In this figure the sampled observations are marked by red points. As expected, the CUSUM filter samples observations more frequently if their deviation to the mean of the time series is relatively high, since the threshold is exceeded more often. Therefore, it is easy to observe that values around the mean are less often sampled than the spikes in the series.

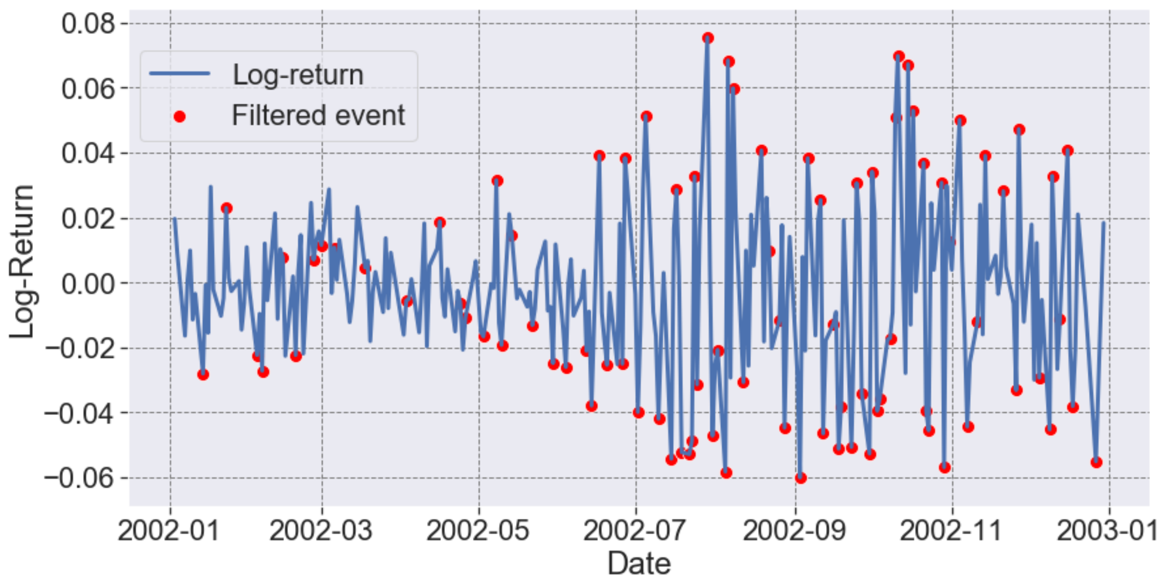


Figure 2.2: Applied CUSUM filter

Once the CUSUM filter is applied, machine learning algorithms can determine whether the occurrence of such events constitutes actionable intelligence. Finally, the choice of h is arbitrary. López De Prado (2018) specifies some choices for h. For the empirical analyis h equals to twice the standard deviation of the DAX log-returns.

2.3 Labelling Financial Data

2.3.1 Triple-Barrier Method

In this chapter the triple-barrier method is explained and the main characteristics of this labelling methodology are demonstrated. Additionally, other methods for labelling are briefly described in order to compare them with the triple-barrier method.
To apply machine learning methods, data has to be in an adequate form to receive information from it. *Bars* can be roughly classified as rows in a data table:

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdot & x_{1,p} \\ x_{2,1} & x_{2,2} & \cdot & x_{2,p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{T,1} & x_{T,2} & \cdot & x_{T,p} \end{pmatrix}$$

The standard bars in financial literature are time bars. This data mostly contains an open, low, high and close price. Hence, one column in X can contain the log-returns of the daily close prices. Other features can be added through feature engineering. Once a matrix X with relevant features is produced out of an unstructural dataset, supervised learning algorithms require the row of X associated with labels or values y. Since in this thesis the algorithm should learn to buy or sell a certain asset, returns has to be classified according to this principle. An often used methodology in the investment literature to classify returns is the fixed-time horizon method. For instance, a CUSUM filter can sample a feature matrix X with I rows, $\{X_i\}_{i=1,\dots,I}$ drawn from some bars with index $t = 1, \dots, T$, where $I \leq T$. Then every row X_i can be associated with a label y_i by

$$y_i = \begin{cases} -1, & \text{if } R^{\text{net}}_{t_{i,0},t_{i,0}+l} < -\kappa \\ 0, & \text{if } |R^{\text{net}}_{t_{i,0},t_{i,0}+l}| \leq \kappa \\ 1, & \text{if } R^{\text{net}}_{t_{i,0},t_{i,0}+l} > \kappa \end{cases}$$

where the net return $R^{\text{net}}_{t_{i,0},t_{i,0}+l}$ over a bar horizon l is defined by

$$R^{\text{net}}_{t_{i,0},t_{i,0}+l} = \frac{P_{t_{i,0}+l}}{P_{t_{i,0}}} - 1.$$

P_t represents the price at time t, whereas parameter κ within the fixed-time horizon method represents a pre-defined threshold value. Although this methodology is very popular in the investment literature, it reveals a main weakness. The same threshold is applied regardless of the observed volatility. Since one of the main characteristics of financial data are volatility clusters, it undersamples observations in low volatility periods and oversamples observations in periods with high volatility (López De Prado 2018).

In this thesis the thresholds should be set dynamically. In order to compute dynamic thresholds, volatility can be modelled by an exponentially weighted moving standard deviation:

$$\sigma_T^2(\text{EWMA}) = \lambda\sigma_{T-1}^2 + (1-\lambda)r_{T-1}^2.$$

The EWMA model captures the dynamic characterisitcs of volatility by applying weighting factors such that the influence of older timepoints decreases exponentially, but never reaches zero. The upper equation holds, if the returns mean is assumed as zero. The smoothing parameter λ controls the decay of the series. A higher λ indicates slower decay while a lower λ indicates faster decay. In the data analysis of this thesis a span of 20 was applied to this formula. The decay parameter λ results from this span by

$$\lambda = \frac{2}{\text{span}+1}, \quad \text{for span} \geq 1.$$

Hence, this approach leads to dynamic volatility estimates which can be used for the triple-barrier method (Zangari 1996).

Now the labelling methodology triple-barrier method is explained. The name of the triple-barrier method is based on the procedure to label observations according to the first barrier touched out of three barriers. The vertical barrier is pre-defined by the number of past bars since the position was taken. The two horizontal barriers represent the trading orders stop-loss limit and profit-taking. The distance between the individual positions and these barriers results directly from the EWMA approach. Hence, these distances are dynamic. If the upper barrier is touched first the observation is labelled as 1 and if the lower barrier is hit first the observation is labelled as -1. Furthermore, if neither of these two barriers is touched, the vertical barrier is broken. Then based on personal preference, the observation is labelled 0 or it will labelled based on the returns sign. In this thesis the observation is labelled as the return sign which causes a binary classification problem. The triple-barrier method is illustrated in Figure 2.3:

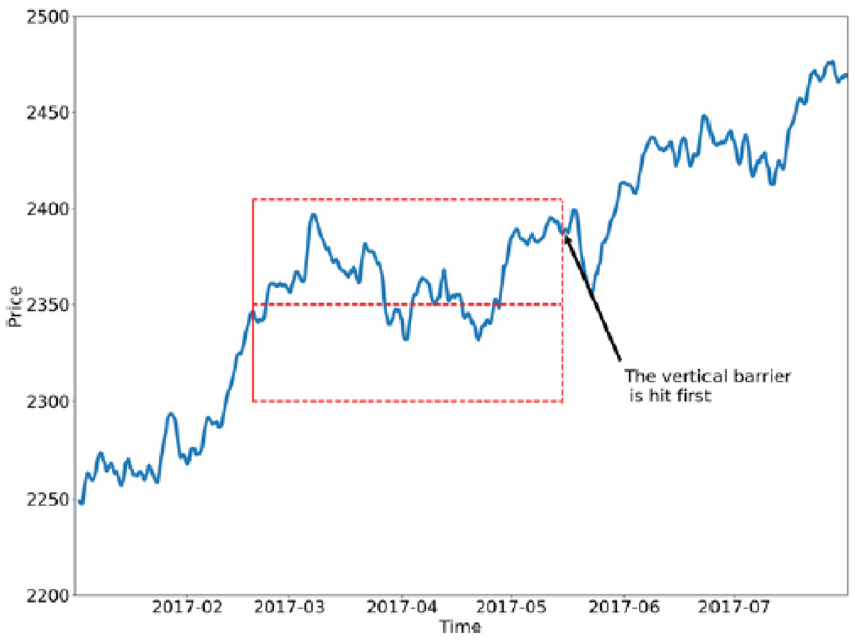


Figure 2.3: Triple-Barrier Method (López De Prado 2018, p.47)

In this graphic the vertical barrier is touched first and the observation is labelled as 1. Since the triple-barrier method is path dependent, the entire path spanning $[t_{i,0}, t_{i,0+l}]$ has to be explored, where l defines the vertical barrier. In the following $[t_{i,1}]$ represents the time of the first barrier hit and the return associated with this label is denoted by $R^{\text{net}}_{t_{i,0},t_{i,1}}$ (López De Prado 2018). As a part of this application, it may happen that labels overlap. Methods dealing with these characteristics will be explained in subsequent chapters.

2.3.2 Meta-Labelling

Before describing a solution to deal with dependent labels, the meta-labelling concept is expounded. While setting a profit-taking and a stop-loss barrier, the triple-barrier method implies a *side* methodology by learning to buy or sell an asset. This approach is applied to a dataset and some learning algorithm concludes whether to buy or sell an asset. However, an investor is also interested in the amount of money he or she should risk in such a bet. Meta-labelling provides a solution for this issue. By learning the side of an asset, feasible labels are $y_i \in \{-1, 1\}$. Now the algorithm should learn whether to take the bet ($y_i = 1$) or to pass it ($y_i = 0$). Hence, $y_i \in \{0, 1\}$ are appropriate outcomes in order to compute a probability to derive the size of a bet, when the side of the bet has already been determined by a primary model (López De Prado 2018).

To describe the effect of meta-labelling, explaining the properties of binary classifier outcomes is helpful. The confusion matrix A.1 illustrates the classification of binary outcomes. This error matrix divides observations into two groups of items. The observations in the left rectangle are called the positives, which exhibit a certain condition and the other group is called negatives which does not exhibit this condition. A binary classifier (bordered blue) tries to predict the obseervations class by their features. The true positives (TP) are correctly predicted positives whereas false positives (FP) are misclassified as positives. As stated in subsection 4.1.3, *precision* measures how many positive classified examples are really positive while *recall* is the share of true positives among the positives. Hence, recall does not contain any information about false positives while precision does not include information about false negatives. For this reason neither precision nor recall provides a complete evaluation of a performance algorithm. The combination of precision and recall about the harmonic mean is called the F1-score (see subsection 4.1.3). The value range of the F1-score is between one and zero. One means that the classifier predicts all items correctly, such that recall and precision equals to one. In general, achieving high precision and recall simultaneously is desired, but this is difficult since increasing the true positives usually comes at a cost of increasing the false negative area (Zhou 2012).

Considering the structure of the F1-measure, meta-labelling can help to increase this classification score. According to López De Prado (2018), the first step is to fit a primary model such that high recall is achieved, even if the precision is not particularly high. Afterwards a secondary model is fitted to correct for the low precision. For this purpose, meta-labelling is applied to the positive predicted by the primary model. Hence, meta-labelling increases the F1-score by filtering out the false positives, whereas the primary model has already identified most of the positive observations. Overall meta-labelling is, as previously mentioned, a tool to learn the *size* of the bet. Therefore, the secondary machine learning algorithm should learn to recognize whether to take the bet or to pass. The “confidence” of the bet is then determined by the prediction probability. Thus, the investor should take a bet if a high probability is predicted for the model outcome and otherwise pass. In general, the classification problem shifts from $y_i \in \{-1, 1\}$ (sell, buy) to $y_i \in \{0, 1\}$ (take bet, pass bet). “Take bet” means that the investor either buys or sells the observed asset. To perform this shift, the primary model has to predict the side of the model $\{-1, 1\}$, then the predicted labels are compared with the true labels. If the prediction was correct $y_i^\star$ is labelled as one, otherwise as zero. The following matrix gives an example about this procedure:

y_i	$\hat{y}_i^{\text{prim}}$		$y_i^\star$
1	1	$\Rightarrow$	1
-1	-1		1
1	-1		0
-1	1		0

Table 2.1: Meta-labelling

Overall, applying meta-labelling leads to a two stage modelling framework. First the primary model has to be fitted on the training data and predict the side on training and test data. Then the whole data receives new labels with respect to meta-labelling. Afterwards, the secondary model is fitted on the training data in order to learn the patterns between features and labels. After this procedure, the secondary model predicts the labels on the features of the test data. Both the predictions of the primary model and the predictions of the secondary model are compared with the true values and their performance is derived with measures explained above and stated in subsection 4.1.3.

2.4 Sample Weights and Overlapping Outcomes

The previous chapter presented the triple-barrier method as a new methodology to label financial observations to provide a useful approach for quantitative investment strategies.

However, the triple-barrier method leads to overlapping outcomes, whenever y_i and y_j depend on a common return. Consider y_i was a function of price bars that occurred over an interval $[t_{i,0}, t_{i,1}]$, whereby $t_{i,1} > t_{j,0}$ and $i < j$. Thus, y_i and y_j will both depend on a common return $R^{\text{net}}_{t_{j,0},\min\{t_{i,1},t_{j,1}\}}$ over the interval $[t_{j,0}, t_{j,1}]$. In the context of the triple-barrier method there are three possible occurrences that a label $y_i = f([t_{i,0}, t_{i,1}])$ overlaps with another label y_j:

1. $t_{j,0} \leq t_{i,0} \leq t_{j,1}$
2. $t_{j,0} \leq t_{i,1} \leq t_{j,1}$
3. $t_{i,0} \leq t_{j,0} \leq t_{j,1} \leq t_{i,1}$

As in most financial time series, these outcomes are not independent. Nonetheless, most machine learning literature is based on independent and identically distributed data. An exception are deep learning methods like convolutional neural networks. Hence, applying machine learning algorithms to IID data leads to better models and more realistic results (López De Prado 2018).

The following content is about explaining methods which tackle the problem of non-IID labels. As stated previously, two labels y_i and y_j are concurrent at t when both are function of a common return $R^{\text{net}}_{t-1,t}$. A general measure of overlapping can be derived for every label y_i by calculating the number of concurrent labels for each y_i. For this approach the number of labels that are common for a given return $R^{\text{net}}_{t-1,t}$ are computed. For each time point $t = 1, \ldots, T$ a binary array $\{1_{t,i}\}_{i=1,\ldots,I}$ is formed where $1_{t,i} \in \{0, 1\}$. The binary variable equals to one if and only if $[t_{i,0}, t_{i,1}]$ overlaps with $[t-1, t]$ and $1_{t,i} = 0$ otherwise. The time intervals $\{[t_{i,0}, t_{i,1}]\}$ are **not** equal since they are created by the triple-barrier method. After calculating the binary array $\{1_{t,i}\}_{i=1,\ldots,I}$, the number of labels concurrent at time point t are computed by $c_t = \sum_{i=1}^{I} 1_{t,i}$ (López De Prado 2018).

The example in Figure 2.4 describes this approach more precisely. The events origin and its duration is visualized through the barrier matrix. Every row can be assigned to one label y_i. Also each label has its own binary array in order to compute the number of concurrent labels at time point t.

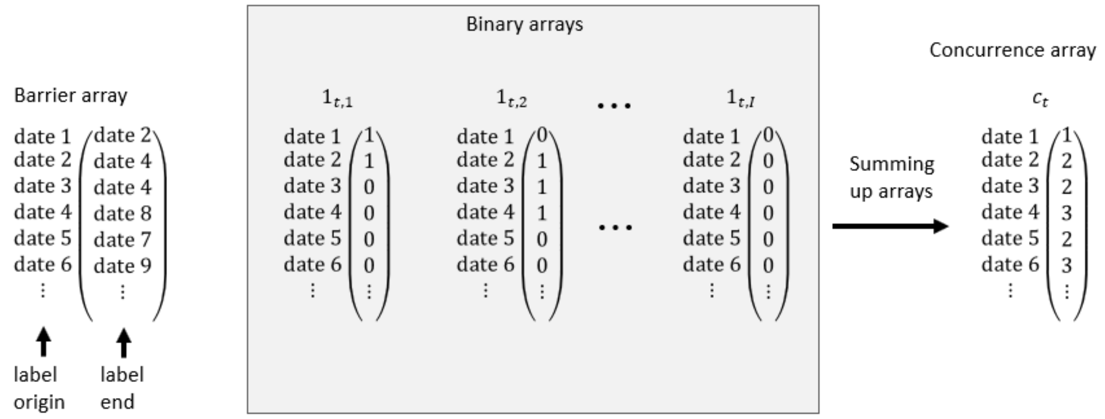


Figure 2.4: Example of concurrent label computing

The equation $c_t = \sum_{i=1}^{I} 1_{t,i}$ is important to calculate the average uniqueness of a label. The average uniqueness is defined by the label's uniqueness (non-overlap) over its lifespan. Then the uniqueness of a label i at time t is

$$u_{t,i} = 1_{t,i} c_t^{-1}.$$

Furthermore, the average uniqueness is defined by

$$\overline{u}_i = (\sum_{i=1}^{T} u_{t,i})(\sum_{t=1}^{T} 1_{t,i})^{-1}.$$

The average uniqueness is helpful to determine the dependency of the labels and can be used to reduce the undue influence of outcomes that contain redundant information. In general, the $\{\overline{u}_i\}_{i=1,\ldots,I}$ needs information that happens in future time. However, this is not problematic since the average uniqueness is not considered as a feature for predictive modelling. Finally, the average uniqueness allows to apply a uniqueness score to a prospective label which is useful for ensemble methods like random forest or bagging (López De Prado 2018).

Figure 2.5 gives an overview of uniqueness values computed on daily DAX data after applying the CUSUM filter method to it. It can be seen that most values vary around 0.18.

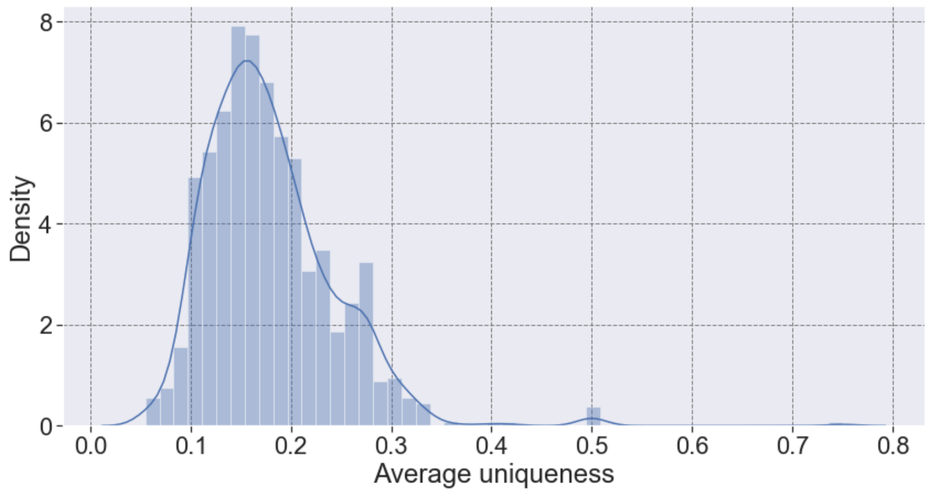


Figure 2.5: Histogram of average uniqueness values

After explaining the computation of average uniqueness, the purpose of considering this computation for bagging classifiers is decribed in the following. To clarify this step, bagging is briefly described. Bagging is an ensemble method which tries to reduce the variance of a model by averaging model outcomes by using, for example, several decision trees. These decision trees are used on a bootstrapped version of the recent dataset to reduce the overall variance. By using a bootstrapped version of a dataset the probability to draw a certain observation i after I draws is

$$\begin{aligned}\mathbb{P}[i \in \text{bootstrap sample}] &= 1 - (1 - \frac{1}{I})^I \\ &\approx 1 - \exp(-1) \\ &= 0.632.\end{aligned}$$

Note that the above equation is true if the sample size grows. This leads to $\lim_{I\to\infty}(1 - \frac{1}{I})^I = 1 - \exp(-1)$. Hence, the expected number of unique observations amounts to 0.632 (Hastie *et al.* 2009).

However, this is not true if the dataset includes overlapping outcomes. Consider $\xi \leq I$ as the number of non-overlapping outcomes in the dataset. Then the equation above changes to

$$\begin{aligned}\mathbb{P}[i \in \text{bootstrap sample}] &= 1 - (1 - \frac{1}{\xi})^{I} \\ &\leq 1 - (1 - \frac{1}{I})^{I\frac{\xi}{I}} \\ &\approx 1 - \exp(-\frac{\xi}{I}) \\ &\leq 1 - \exp(-1).\end{aligned}$$

Hence, the expected number of sampled unique observations are lower than in the non-overlapping case if $\xi < I$ (López De Prado 2018).
Considering bagging as a classifier on observations with $I^{-1}\sum_{i=1}^{I}\overline{u}_i << 1$ leads to an increasing probability of observations to be redundant to each other. Additionally, the observations are similar to the out-of-bag observations. Solving the first issue with $I^{-1}\sum_{i=1}^{I}\overline{u}_i << 1$ can be performed by sampling only a fraction of the observations which is predefined by utilizing the average uniqueness. Instead of drawing I observations, only $I^{\star} = I^{-1}\sum_{i=1}^{I}\overline{u}_i$ are considered. Therefore, the in-bag observations are sampled according to their average uniqueness such that they are not sampled at a frequency higher than their uniqueness (López De Prado 2018).
As mentioned previously, computing average uniqueness on a dataset is a tool to make bootstrap samples more independent and identically distributed. Furthermore, observations can be weighted by their return attribution. Obviously, higher absolute returns should have a higher weight in the fitting procedure of machine learning methods, because higher absolute returns have a bigger impact on the prospective portfolio return. Additionally, the uniqueness of label impacts the weight of a label since drawing unique values is preferred. The following equations provide an idea of weighting observation returns with respect to their absolute amount over their lifespan.

$$\tilde{w}_i = \left| \sum_{t=t_{i,0}}^{t_{i,1}} \frac{r_{t-1}}{c_t} \right|$$

$$w_i = \tilde{w}_i I \left(\sum_{j=1}^{I} \tilde{w}_j \right)^{-1}$$

These weights are scaled up to the number of filtered observations $\sum_{i=1}^{I} w_i = I$ for computational reasons. Thus, the method weights an observation according to its absolute return that is attributed uniquely to it (López De Prado 2018).

In the following, the main points of this chapter are briefly summarized. Average uniqueness can be utilized to draw an appropriate sample size within a bagging classifier. Moreover, this methodology makes the samples more IID which is useful for applying machine learning methods to it. Another part of affecting sample weights is the return attribution approach, which gives higher absolute returns more impact in the fitting process.

2.5 Bet Sizing

After applying machine learning methods to a financial dataset, probabilities can be derived for each value in the dataset. The obvious question regarding investing leads to appropriate bet sizing or portfolio weights. A machine learning algorithm may achieve superior performance measures, but without adequate bet sizing the investment strategy will lose money. The following content is about mapping probabilities to bet sizes.

Let $p(y)$ be a certain probability that the event y takes place, $Y = \{-1, \dots, 0, \dots, 1\}$ various labels associated with bet sizes and $y \in Y$ the predicted label. For each label $i = 1, \dots, ||Y||$ a probability p_i is estimated, with $\sum_{i=1}^{||Y||} p_i = 1$. To test the null hypothesis $H_0 : \tilde{p} = \frac{1}{||Y||}$, a test statistic

$$\zeta = \frac{\tilde{p} - \frac{1}{||Y||}}{\sqrt{\tilde{p}(1-\tilde{p})}} \sim \Phi,$$

with $\zeta \in [0, \infty)$ can be considered, whereas Φ represents the standard normal distribution. $\tilde{p}$ is defined by the largest probability $\tilde{p} = \max_i\{p_i\}$. Hence, the bet size can be calculated with $b = y(2\Phi(\zeta) - 1)$, where $b \in [-1, 1]$ and $\Phi(.)$ represents the cumulative distribution function (López De Prado 2018).

This approach can be illustrated by the example of meta-labelling $y_i \in \{0, 1\}$. Then, the null hypothesis equals to $H_0 : \tilde{p} = \frac{1}{2}$. For every label in the test data the model predicts two probability values. If $p(y = 0) > p(y = 1)$, $\tilde{p}$ equals to $p(y = 0)$ and if $p(y = 0) < p(y = 1)$, $\tilde{p}$ equals to $p(y = 1)$. Considering $\tilde{p} = p(y = 1)$, the test statistic

ζ gets larger with a larger value of $\tilde{p}$. Consequently, this results in a larger bet size, as illustrated in Figure 2.6 for $y = 1$.

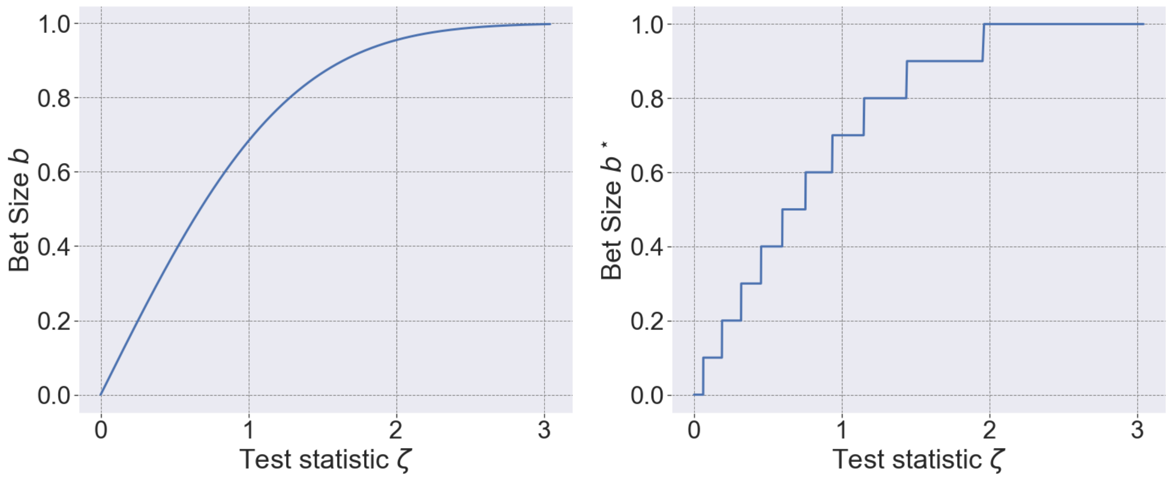


Figure 2.6: Transformation to bet sizes

Considering $y = 0$ ($p(y = 0) > p(y = 1)$), the probability value of $p(y = 0)$ has no impact on the bet size. This is in line with the meta-labelling approach since predicting $y = 0$ suggests not to bet. However, as stated in subsection 2.3.1, the *side* of the primary model must be included in this application. For this reason, the bet size b is multiplicated with either 1 (buy) or -1 (sell). Hence, the y-axis of Figure 2.6 switches its values range from $[0, 1]$ to $[0, -1]$ in the sell case. As described in subsection 2.3.1, every label and consequently every bet is associated with a holding period spanning from the time it originated to the time the first barrier is touched. To consider the applications overlapping characteristic, all sizes across all bets still active at a given point in time are averaged. This reduces some of the excess turnover. As it is still likely that small trades will be triggered with every prediction, the bets are discretized as visualized in Figure 2.3.1. For this reason, the bet size is adjusted with $b^\star = round[\frac{b}{d}]d$, where $d \in [0, 1]$ (López De Prado 2018).

The upper approach leads to bet sizes of $b = 0$. Since this thesis focuses on an investing approach, the bet sizes are treated as portfolio weights. According to subsection 2.3.1, the bet sizes with $b = 0$ are adjusted such that the portfolio remains unchanged. Thus, the previous portfolio value is assigned to the bet sizes with $b = 0$.

3 Machine Learning Applications in Finance

3.1 Sources of Errors in Machine Learning Algorithms

In this section the machine learning algorithms used in the empirical analysis are explained. Before describing the learning algorithms more precisely, the bias-variance decomposition will be clarified and investigated.

Considering training observations $\{x_t\}_{t=1,...,T}$ and real valued labels $\{y_t\}_{t=1,...,T}$, a function $Y = f(x) + \epsilon$ is assumed, where $\mathbb{E}[\epsilon] = 0$ and $\text{Var}[\epsilon] = \sigma_\epsilon^2$ holds. Target of the later illustrated learning algorithms is to find a function $\hat{f}(x)$ which fits $f(x)$ as well as possible. Therefore, the prediction error of the fitted function $\hat{f}(x)$ should be minimal (Hastie *et al.* 2009).

An expression of the prediction error to a new input x_0 can be derived by the usage of the mean squared error to obtain the variance-bias decomposition. This results in the variance-bias decomposition as stated by Hyndman (2015) (cf. proof in A.1)

$$
\begin{aligned}
\mathbb{E}[(y - \hat{f}(x_0))^2] &= \sigma_\epsilon^2 + \mathbb{E}[\hat{f}(x_0) - f(x_0)]^2 + (\mathbb{E}[\hat{f}(x_0)] - \mathbb{E}[\hat{f}(x_0)])^2 \\
&= \sigma_\epsilon^2 + \text{Bias}^2(\hat{f}(x_0)) + \text{Var}[\hat{f}(x_0)] \\
&= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance}
\end{aligned}
$$

According to Manning *et al.* (2009), the upper equation can be transferred to a classification problem, although it is suited for regression problems. To summarize, the mean squared error of a new observation consists of the irreducible error, the squared bias and the variance. The *noise* term σ_ϵ^2 represents the variance of the target around the true mean $f(x_0)$ and cannot be explained by any model. In addition, the bias is defined by the amount of difference between the average of the estimates and the true mean. When the bias is high, the algorithm has failed to recognize important patterns, hence the model is underfitted. As a last point the variance of the estimator $\hat{f}(x_0)$ is defined by the expected squared deviation around its mean. When the variance is high, the algorithm has mistaken noise for signal, instead of learning the general patterns in the training data. This phenomenon is called overfitting. As a consequence the model produces wildly different outputs for minimal changes in the training set (López De Prado 2018).

3.2 Logistic Regression

Facing a binary target variable considering logistic regression is a popular model choice to fit a model. For the model approach T observations of the form of $y_t, x_{t,1}, \ldots, x_{T,p}, t = 1, \ldots, T$ where $y_t \in \{0, 1\}$ are considered. The main goal of the binary regression is to model the features impact on the conditional probability

$$\pi_t = \mathbb{P}(y_t = 1 | x_{t,1}, \ldots, x_{t,p}) = \mathbb{E}[y_t | x_{t,1}, \ldots, x_{t,p}]$$

for the occurrence of the target variable y_t given the features $x_{t,1}, \ldots, x_{t,p}$. One model assumption is to assume conditional independence of the target variable. For this reason, the maximum likelihood can be calculated as a product. As mentioned before, this can be a real issue in the DAX time series because the y_t are not independent. To receive a binary output the outcome of the linear predictor

$$\eta_t = \theta_0 + \theta_1 x_{t,1} + \cdots + \theta_p x_{t,p}$$

with parameters $(\theta_0, \theta_1, \ldots, \theta_p)$ has to be in the interval $[0, 1]$ (Fahrmeir *et al.* 2009).

Hence, a transformation function for η_t is required. The sigmoid function $\phi(\eta) = \frac{1}{1+exp(-\eta)} = \pi_t$ maps η to the required range. Maximizing the loglikelihood $\sum_{t=1}^{T} y_t \text{log}(\phi(\eta_i)) + (1 - y_t)\text{log}(1 - \phi(\eta_i))$ leads to the desired θ values. As stated by Raschka & Mirjalili (2017), this can be also expressed by minimizing a penalty function

$$J(\theta) = \left[\sum_{t=1}^{T} y_t(-\text{log}(\phi(\eta_i))) + (1 - y_t)(-\text{log}(1 - \phi(\eta_i))) \right] + \frac{1}{2C} ||\theta||^2.$$

One opportunity to find a good bias-variance trade-off consists of reducing model complexity by regularization. Penalizing to high θ values is a useful tool to deal with collinearity and overfitting. In the upper equation L2 loss is represented by $||\theta||^2$. This can be replaced by L1 loss $||\theta||$, considering absolute loss instead of squared loss. C represents the regularization strength in the equation. A decreasing of C leads to a higher regularization overall. In contrast to decision trees, the logistic regression is in general not suited to constitute non-linear patterns between features and the target variable.

3.3 Classification Trees

Classification trees are a popular tool to detect patterns in a dataset. Introduced by Breiman *et al.* (1984), a CART (classification and regression tree) constructs binary splits top-down. The associated split points are created by a pre-defined split criterion which is applied to the feature space $\mathcal{X}$. According to Hastie *et al.* (2009), for a label $y_t \in \{0, 1\}$ with corresponding p features $x_t = x_{t,1}, \ldots, x_{t,p}$ for $t = 1, \ldots, T$ the model output can be computed by

$$f(x) = \sum_{m=1}^{M} \hat{O}_m(x) \chi_{\mathcal{R}_m}(x).$$

Thus, the tree divides the feature space $\mathcal{X}$ into M rectangles $\mathcal{R}_m \in \mathcal{X}$, where $\hat{O}_m(x)$ is a predicted numerical response, class label or class distribution output. In addition, $\chi_{\mathcal{R}_m}(x)$ represents the indicator funtion.
To receive an optimal split, a greedy algorithm called “exhausted search” is used. This procedure searches the best splitting feature and its corresponding optimal split point by minimizing a split criterion to divide the dataset into two half-spaces for a split point v and a feature j

$$\mathcal{R}_1(j, v) = \{X | X_j \leq v\} \quad \text{and} \quad \mathcal{R}_2(j, v) = \{X | X_j > v\}.$$

These half-spaces are divided repeatedly into half-spaces until, for instance, only a pre-defined amount of observations is left in a half-space. In the case of classification trees, the algorithm tries to separate the feature space $\mathcal{X}$, such that homogeneous half-spaces are created (James *et al.* 2013).
For finding the split points, the resulting half-spaces are compared with the original dataset by an impurity measure $\mathcal{I}(\mathcal{R})$. Consider $\mathcal{R}$ as a parent node with two child nodes $\mathcal{R}_1$ and $\mathcal{R}_2$. If impurity can be quantified in each half-space, potential splits in a node $\mathcal{R}$ can be evaluated by

$$\mathcal{I}(\mathcal{R}) - \frac{|\mathcal{R}_1|}{|R|}\mathcal{I}(\mathcal{R}_1) - \frac{|\mathcal{R}_2|}{|\mathcal{R}|}\mathcal{I}(\mathcal{R}_2),$$

whereas $|\mathcal{R}|$ represents the number of datapoints in node $\mathcal{R}$ (James *et al.* 2013). The splitting criteria $\mathcal{I}(\mathcal{R})$ for classification trees can be one of the following:

$$\begin{aligned} \text{Missclassification Error} &: 1 - \hat{p}_{mq} \\ \text{Gini Index} &: \sum_{q=1}^{Q} \hat{p}_{mq}(1 - \hat{p}_{mq}) \\ \text{Entropy} &: -\sum_{q=1}^{Q} \hat{p}_{mq}\text{log}(\hat{p}_{mq}) \end{aligned}$$

$\hat{p}_{mq}$ corresponds to the relative frequency of category q in the response node m

$$\hat{p}_{mq} = 1/T_m \sum_{x_t \in \mathcal{R}_m} \chi_q(y_t),$$

whereas $T_m = \#\{x_t \in \mathcal{R}_m\}$ (Hastie *et al.* 2009).
In general, these three impurity measures are quite similar. However, the entropy and gini index criterion are more sensitive to changes in the node probabilities p. In the data analysis part the entropy measure is considered to split decision trees. In the case of two classes ($y_t \in \{0, 1\}$) the formula for entropy can expressed through $-\sum_{q=1}^{2} \hat{p}_{mq}\text{log } \hat{p}_{mq} = -p \text{ log}(p) - (1 - p)\text{log}(1 - p)$, where p repressent the fraction of one class and the basis for log is equal to the number of classes in y_t (Hastie *et al.* 2009).

As mentioned previously, the exhaustive search algorithm tries to separate the respective dataset into two homogeneous datasets. This coincides with the entropy feature because the entropy measure is constructed in a way that it has a maximum at $p = 0.5$. This represents a half-space with maximum heterogeneity where half of the observations belongs to one class and the other half to the other class. In this case the algorithm would if possible choose another split point. Another advantage of decision trees is their interpretability. In contrast to other machine learning methods, like boosting or support vector machine, the decision tree fitting procedure can be understood with the help of its visualization. In Figure 3.1 an example of a decision tree is illustrated. The nodes are represented by boxes. A box with a majority of positive labelled classes ($y_t = 1$) is blue, whereas the opposite case is red. The nodes with a darker blue contain a greater majority with positive labelled classes than the boxes with a ligher blue. For this reason, the darker boxes has also a lower entropy. Furthermore, according to López De Prado (2018), labels are balanced regarding their relative frequency to prevent that trees or logistic regression misclassify minority classes.

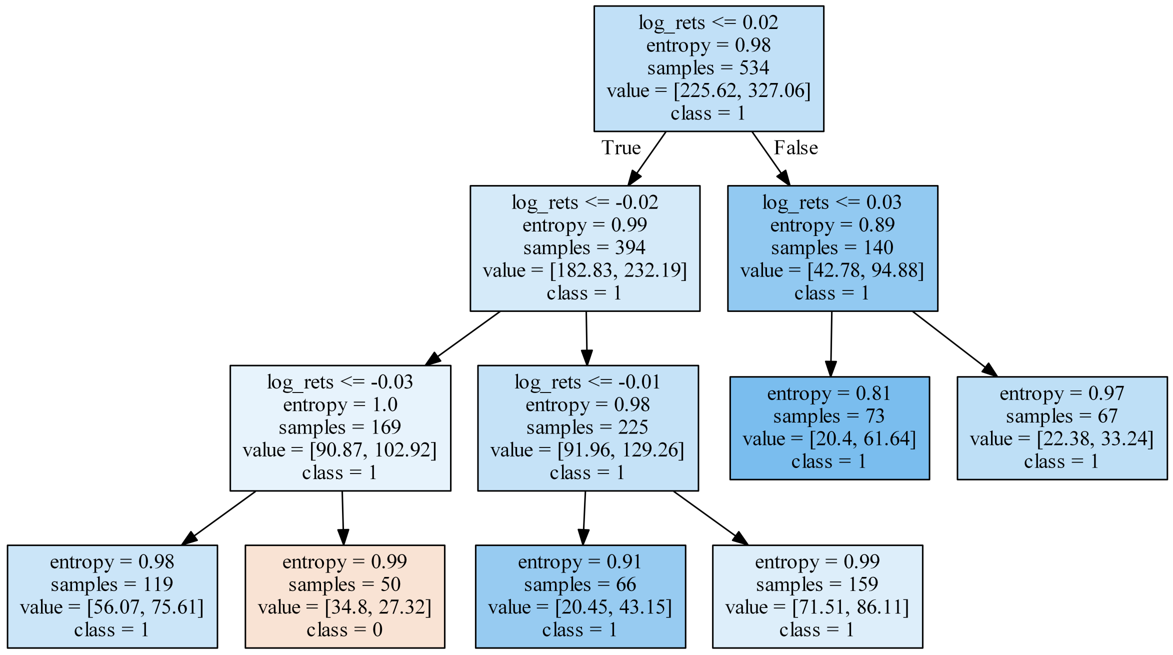


Figure 3.1: Example of a decision tree

3.4 Bagging

The following content is about ensemble methods used in the empirical analysis. The first model explained is the bootstrap aggregation (*bagging*) classifier, introduced by Breiman (1996). In general, decision trees suffer from high variance. This means, by adding or removing an observation the tree structure can change rapidly. In contrast, low variance methods like linear regression yield rather similar results if applied repeatedly to distinct datasets. Utilizing bootstrap aggregation on a statistical learning method can reduce the variance of the considered model (James *et al.* 2013).
Consider, for instance, independent observations $X_1, X_2, \ldots, X_B$ (each with variance

σ^2) and compute the average of the observations $\overline{X}$. Then the variance of the average is given by σ^2/B. Hence, the variance of $\overline{X}$ can be reduced by adding further independent observations. Thus, one possibility would be to take as many trainings sets as possible, build a separate training model for each training set and average the resulting predictions. However, this is not practical because multiple training sets are in general not available. As an alternative, bootstrapping can be applied to one dataset to generate B different sets of observations. With the use of bootstrapping, an arbitrary model is applied to each of those training sets to receive $\hat{f}^{\star,1}(x), \dots, \hat{f}^{\star,B}(x)$ different models. Hence, the predictions of these models are averaged:

$$\hat{f}^{\text{bag}}(x) = \frac{1}{B}\sum_{i=1}^{B}\hat{f}^{\star,i}(x).$$

In classification tasks a majority vote of the B classifiers leads to the predicted class (James *et al.* 2013).
As mentioned, bagging's main advantage is the forecasts variance reduction. Consider $\hat{f}^{\star,i}(x)$ as one prediction of B estimators used in the bagging application and σ^2 and ρ as the average single variance and the average correlation of an estimator respectively. In the lower equation, $\sigma_{i,j} = \text{Cov}(\hat{f}^{\star,i}(x), \hat{f}^{\star,j}(x))$ represents the covariance of predictions and $\rho = \text{Corr}(\hat{f}^{\star,i}(x), \hat{f}^{\star,j}(x))$ the correlation. Then the variance of the predictions can be derived as:

$$\begin{aligned}
\text{Var}\left[\frac{1}{B}\sum_{i=1}^{B}\hat{f}^{\star,i}(x)\right] &= \frac{1}{B^2}\left(\sum_{i=1}^{B}\text{Var}(\hat{f}^{\star,i}(x)) + \sum_{i=1}^{B}\sum_{i\neq j=1}^{B}\sigma_{i,j}\right)\\
&= \frac{1}{B^2}\left(B\sigma^2 + \sum_{i=1}^{B}\sum_{i\neq j=1}^{B}\frac{\sigma_{i,j}}{\sqrt{\sigma^2}\sqrt{\sigma^2}}\sigma^2\right)\\
&= \frac{1}{B^2}\left(B\sigma^2 + \underbrace{\sum_{i=1}^{B}\sum_{i\neq j=1}^{B}}_{\substack{B(B-1)\\ \text{summands}}}\rho\sigma^2\right)\\
&= \frac{1}{B^2}(B\sigma^2 + B(B-1)\rho\sigma^2) = \frac{\sigma^2}{B}(1 + (B-1)\rho)\\
&= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2
\end{aligned}$$

Therefore, the effectiveness of bagging depends on the correlation among their forecasts

ρ, since $\rho \to 1 \Rightarrow \text{Var}[\frac{1}{B}\sum_{i=1}^{B} \hat{f}^{\star,i}(x)] \to \sigma^2$. Thus, as already mentioned in subsection 2.4, producing samples independent samples is very important for bagging. In the worst case applying bagging is useless regarding the final predictor's variance. Setting the maximal fraction of samples according to the average uniqueness of a label reduces the correlation among the estimators and subsequently lowers the variance of the forecasts. If, for example, each observation at t is labelled according to the return between t and $t + 100$, only 1% of the observations should be sampled per bagged estimator in order to make the drawn samples more independent. For this purpose, the number of observations drawn in the bootstrap approach is reduced to the mean average uniqueness of the labels $I^\star$ (López De Prado 2018).

Overall, bagging is a decent method to reduce the variance of a weak learner like trees. However, it is not a classifier which reduces the bias of an already weak classifier.

3.5 Random Forest

3.5.1 Model Description

As introduced by Breiman (2001), random forest application shares similarities with the previously mentioned ensemble model bagging. In general, decision trees are perfect for bagging and random forests because they can identify complex patterns in a dataset and have low bias if grown sufficiently deep. Random forest is similar to bagging since the method builds a large collection of de-correlated trees and then average them. For this purpose, bootstrapping the considered feature matrix X is required to reduce the variance of fitted decision trees. An average variance of B IID random variables is given by $\frac{1}{B}\sigma^2$. As mentioned in subsection 3.4, this formula changes for identically distributed but not independet data to

$$\sigma^2\left(\rho + \frac{1-\rho}{B}\right).$$

The core difference between bagging and random forest is the way of sampling columns of the feature matrix X. In bagging all features are used for each decision tree classifier. In contrast to bagging, random forests consider only a fraction of features to build the trees. Thus, the variance of the prediction is reduced by reducing the correlations between the trees (Hastie *et al.* 2009). According to subsection 2.4, the random forest is slightly modified. In contrast to sampling observations equal to the number of rows in the dataset, the number of observations equals to the the mean average uniqueness of the labels $I^\star$.

The Algorithm 1, as stated by Hastie *et al.* (2009), describes the random forest formally:

Algorithm 1 Random Forest for Classification

```
1. For $i = 1, \dots B$
    (a) Draw a bootstrap sample of size $I^\star$ from the training data
    (b) Grow a random forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size or the maximum depth is reached.
        (i) Select $\sqrt{p}$ variables at random from the $p$ variables
        (ii) Pick the best variable/split-point among the $\sqrt{p}$
        (iii) Split the node into two daughter nodes
2. Output the ensemble of trees
```

To make a new prediction at a new point x:
Classification: Let $\hat{O}_i(x)$ be the class prediction of the i-th random forest tree. Then $\hat{O}^B_{\text{RF}}(x) = \text{majority vote}\{\hat{O}_i(x)\}^B_1$

As illustrated in Figure 3.2, the trees are different from each other with respect to the selected features. A new observation x_0 would be pass through all trees and the majority vote would assign the final label to it.

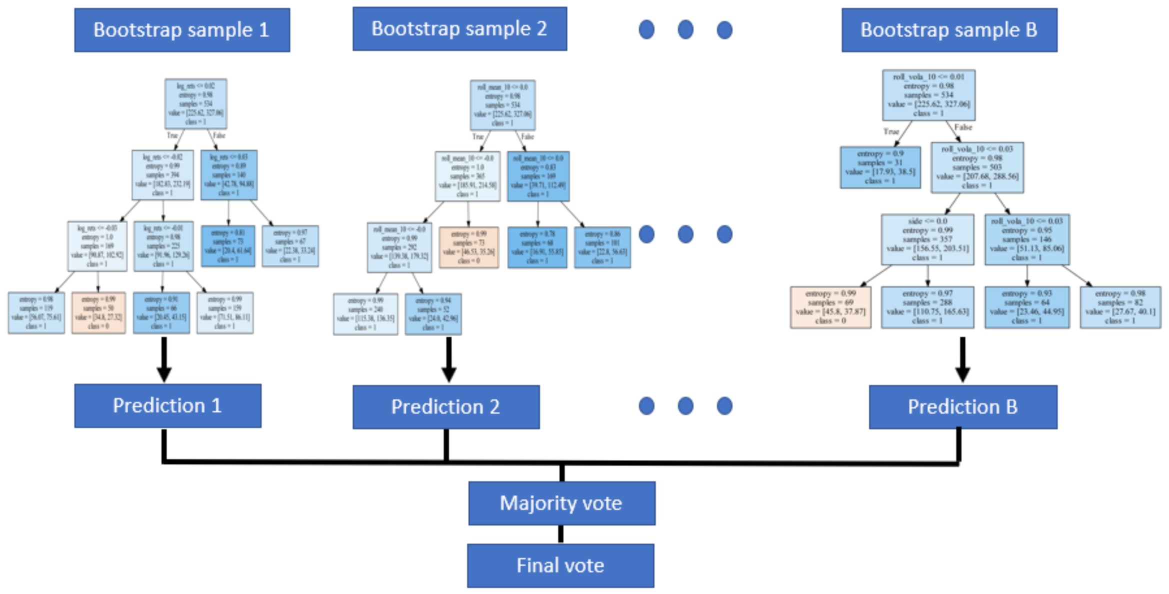


Figure 3.2: Example of random forest

3.5.2 Feature Importance

In general, ensemble methods are a powerful tool to find patterns in datasets. However, the algorithms learn without directing the process to the user (Black Box!). Random forests deliver not only good results in predictive modelling, their special characteristic of modelling provides also the possibility to measure feature importance. According to Breiman (2001), feature importance can be measured by permuting a feature and subsequently calculating the increase in the model's predictions. In the following content three different feature importance methods are described. They mainly differ regarding their dealing with substitutional effects, applying the feature importance approach to in sample data or out of sample data and if the method can be used uniquely to tree-based classifier. Substitution effects take place when estimated importance of one feature is reduced by the presence of another feature. As stated by López De Prado (2018), it is an analogy to multicollinearity.

- Mean Decrease Impurity

As stated by Louppe *et al.* (2013), mean decrease impurity (MDI) is a fast in sample method specific for tree-based classifier. As described earlier in this section, building a decision tree by splitting variables is based on the entropy in the potential tree nodes. The split point is selected such that the impurity in the subsets is decreased. Hence, for buliding a decision tree the overall impurity decrease assigned to each feature can be derived. Facing a forest of trees, these values can be averaged over all trees and ranked accordingly. Additionally, effects are masked if some features are systematically ignored by the decision tree in favour of others. This is prevented by setting the maximum features drawn to one. Consequently, each feature is given a chance to be splitted. Furthermore, MDI does not address substitution effects in the presence of correlated features. Thus, the importance of two identically features will be halved (López De Prado 2018).

- Mean Decrease Accuracy

Mean decrease accuracy (MDA) is a permutation-based feature importance measure introduced by Breiman (2001). Out of sample observations in random forests can be used to measure the prediction strength of each feature. To examine feature importance, the classifier is fitted on features X first. Then it derives its performance scores (accuracy, log-loss, ROC-AUC). After this procedure, only one feature at once, i.e. each column of X, is permuted to derive the performance of the classifier. The decrease in, for example, accuracy as a result of the permutation can be used to derive the feature importance. One key difference to MDI is that feature importance is based on out of sample observations, so it measures the predictive power of features. In addition, this method can be applied to all classifiers, not only to tree-based classifiers. Applying this procedure to logistic regression would only require a proper cross-validation method. In addition to these characteristics, MDA is vulnerable to substitution effects in the presence of correlated features. Considering two identical features, discarding one will not change the predition, even if the feature is critical for predicting (López De Prado 2018).

- Single Feature Importance

The so far introduced feature importance tools suffer both from substitution effects in the data. This could lead to false conclusions when the model must be understood, improved or simplified. Single feature importance (SFI) deals with this problem in a way, that the out of sample score is computed for each feature in isolation. Hence, no substitution effects take place, since only one feature is taken into consideration at a time. On the other hand, the main limitation of SFI is its incapacity to measure dependeny structure between two or more features. For example, feature A may be only useful in connection with feature B, even if feature A is not beneficial alone (López De Prado 2018).

3.6 Cross-Validation and Tuning

3.6.1 Cross-Validation in Finance

Cross-validation (CV) is a widely used statistical methodology. Its purpose is to estimate directly the expected extra-sample error $\text{Err} = \mathbb{E}[L(Y, \hat{f}(X))]$. This is the average generalization error, when $\hat{f}(x)$ is applied to an independent test sample. This methodology can be transferred to the classification case (Hastie *et al.* 2009).

Cross-validation splits observations drawn from an IID process into two datasets: A training set and a test set. For obvious reasons every observation should belong to one dataset, such that no leakage is present. One popular approach is the k-fold CV. With this approach the dataset is divided into k parts, where one part represents the test and the other parts the trainig dataset. A machine learning algorithm is applied to the train partitions and tested on the test partition. Figure 3.3 illustrates the idea of this methodology for $k = 5$.

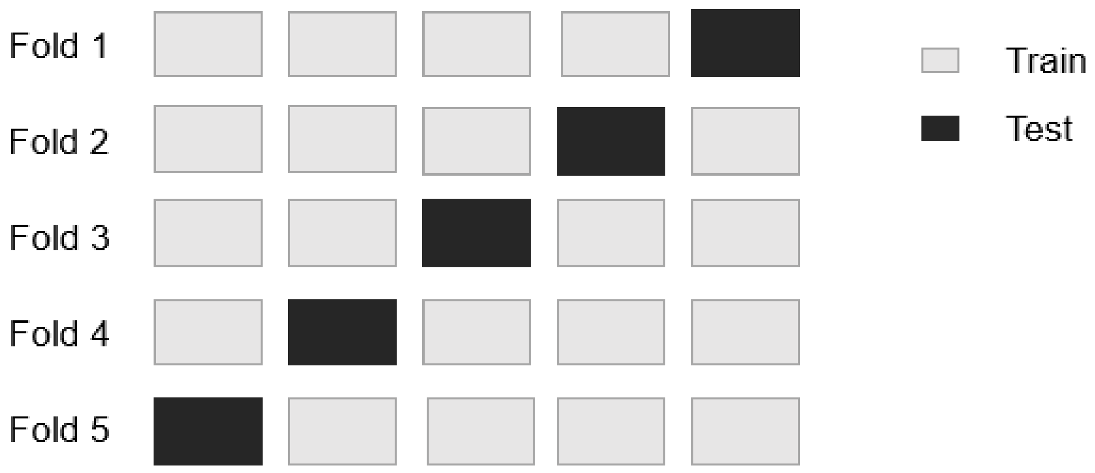


Figure 3.3: Train/test splits in a 5-fold scheme

The splitting process is described in algorithm 2:

Algorithm 2 Cross-Validation

```
1. Dataset is partitioned into $k$ subsets
2. For $i = 1, \dots, k$
    (a) Machine Learning algorithm is trained on all subsets excluding $i$
    (b) Fitted algorithm is tested on $i$
```

After applying this methodology, K different performance measures are received. In the regression case this could be K mean squared error estimates while in the classification case other measures like F1 could be chosen. Hence, the final prediction error can be estimated by

$$\mathrm{CV}(\hat{f}) = \frac{1}{T} \sum_{i=1}^{T} L(y_t, \hat{f}^{-K}(x_i)),$$

where $\hat{f}^{-K}(x_i)$ is the fitted function on all partitions excluding the part K (Hastie *et al.* 2009).

3.6.1.1 Purging

K-fold CV can be very useful in the presence of IID data, but in financial applications observations cannot be assumed to be drawn from an IID process. To be independent, training and test data must be independent. However, especially labelling with the triple-barrier method provides overlapping labels. In general, identical information in the training and test set causes leakage. Hence, with

- Serial correlation, i.e. $X_t \approx X_{t+1}$ and
- Overlapping labels, i.e. $Y_t \approx Y_{t+1}$

in the dataset, information is leaked by placing label t and label $t+1$ in different sets. For instance, a classifier trained on (X_t, Y_t) is asked to predict $\mathbb{E}[Y_{t+1}|X_{t+1}]$. Consequently, the classifier is more likely to achieve $Y_{t+1} = \mathbb{E}[Y_{t+1}|X_{t+1}]$ even if X is an irrelevant feature. In order to prevent the classifier to profit from leakage, overfitting in the machine learning algorithm should be avoided. During bagging and random forest, this can be performed by early stopping the decision tree base estimator. A solution to reduce leakage is called *purging*. This approach purges all observations from the training data whose labels are concurrent (cf. subsection 2.4) with those labels included in the test set. As stated in subsection 2.4, two labels Y_i and Y_j are concurrent, if they share at least one common return. Therefore, to purge the training set appropriately, the different overlapping schemes has to be kept in mind. Whenever one of the three cases defined in subsection 2.4 occurs, the respective observations have to be removed from the training dataset (López De Prado 2018).

3.6.1.2 Embargo

Another solution to deal with leakage is called *embargo*. In this approach an embargo is imposed after every test set. Considering $t_{i,1} < t_{j,0}$, training labels $Y_i = f([t_{i,0}, t_{i,1}])$ contain information which was available at testing time $t_{j,0}$. Hence, the training set prior the test set does not need to be affected. The embargo time e can be implemented by setting $Y_i = f([t_{j,0}, t_{j,1} + e])$ before purging. As recommended by López De Prado (2018), a value of $e = 0.01T$ is used in this thesis for imposing embargo. Figure 3.4 illustrates the embargo approach.

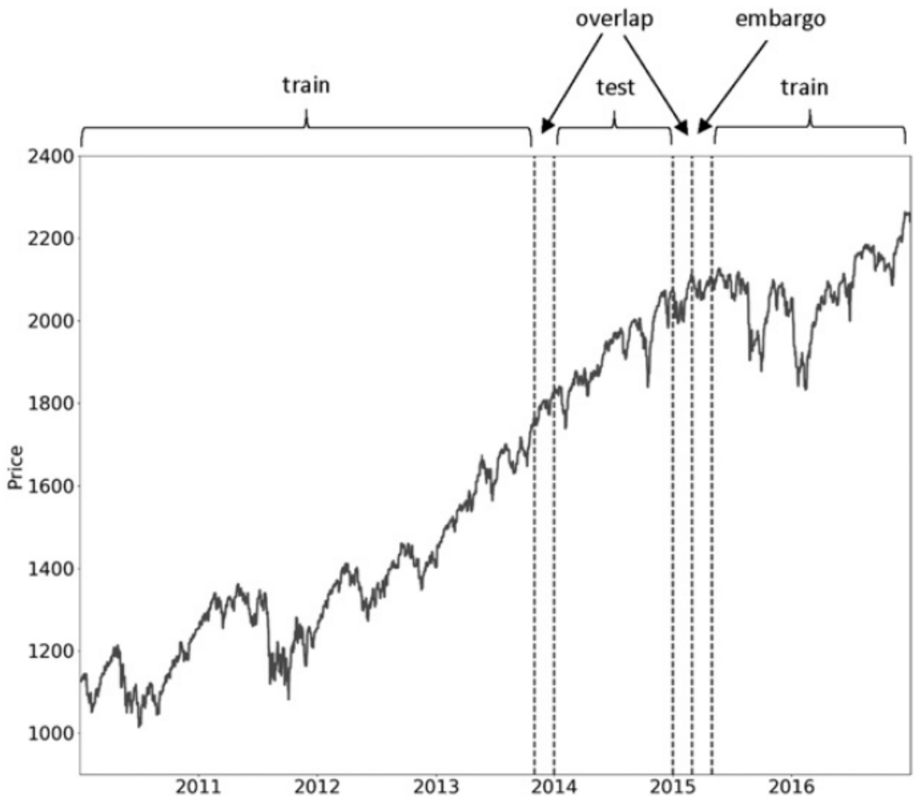


Figure 3.4: Embargo of post-train observations (López De Prado 2018, p.108)

Afterwards, a purged K-fold CV can be derived by purging and imposing embargoing whenever a dataset is splitted into training and test data. Hence, the main pitfall called leakage is reduced/prevented.

3.6.2 Hyperparameter Tuning

In this section hyperparameter tuning is briefly introduced. The so called hyperparameters in machine learning applications are not determined by the usual fitting procedure, but rather pre-defined prior the fitting process. By contrast, for example, the model parameters θ_i are optimized in a linear model $y_t = x_{tp}\theta$ during fitting. Most of the machine learning algorithms have hyperparameters like pre-defined maximum depth for trees. The goal is to optimize these hyperparameters with respect to the estimated prediction error by cross-validation. Hence, tuning can be important to control the capacity of the model regarding overfitting or underfitting. Since overfitting in terms of noise learning is a major issue in analysing financial data, tuning is an important part of controlling the capacity of the prospective model.

One approach to utilize tuning is grid search. At grid search a pre-defined parameter set has to be provided. For example, an optimal choice for decision trees' maximum depth is desired. A provided parameter set can look like

- max depth $\{2, 3, 4, 5, 6\}$
- min weight fraction leaf $\{0.05, 0.1, 0.15\}$,

where "min weight fraction leaf" represents the minimum fraction of observation in a leaf. With the grid search methodology every combination of the upper hyperparameters is evaluated. This can be done by using purged K-fold cross-validation. Thus, the training set is, for example, partitioned into three parts, where one part is the new test set and the other two is the respective training set. After applying every combination to each training set and evaluating the fitted model on the respective test set, the favourite model is determined by the best result. For classification purposes the performance measure might be either ROC-AUC, Recall or F1-measure as described in subsection 4.1.3. Afterwards, the best hyperparameter combination is considered for fitting to the whole training set (López De Prado 2018). The previously mentioned procedure is visualized in Figure 3.5.

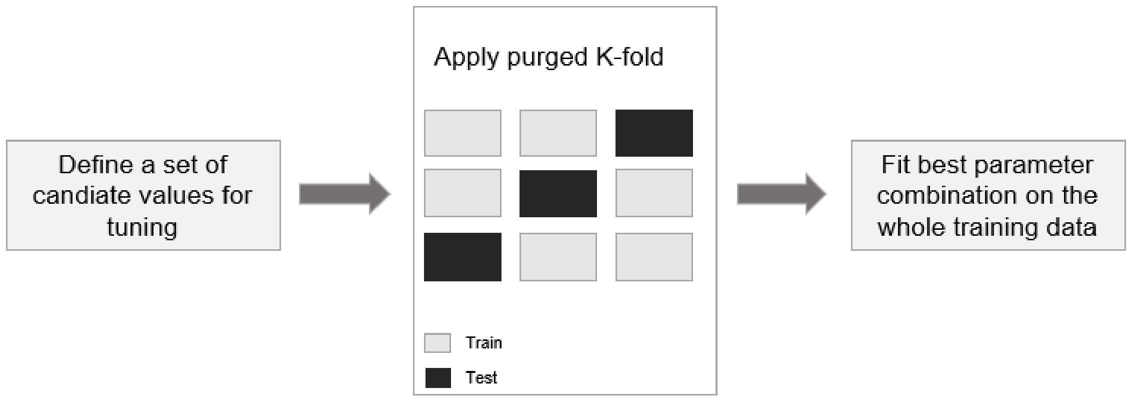

Define a set of candiate values for tuning
Apply purged K-fold
Train
Test
Fit best parameter combination on the whole training data

Figure 3.5: A schematic of the tuning process

4 Backtesting

4.1 Backtest Statistics

To evaluate a potential investment stratgey, several backtest measures can be computed.

4.1.1 Sharpe Ratio

Introduced as a measure for the performance for mutual funds by Sharpe (1966), the *Sharpe ratio*

$$SR = \frac{\mu}{\sigma}$$

is one of the most popular backtest measures. The purpose of the Sharpe ratio is to examine the performance of an investment by adjusting it for its risk. Therefore, the mean μ and standard deviation σ has to be estimated. Thus, the true parameters are not certainly known. For this reason, the returns $\{r_t\}_{t=1,\dots,T}$ are assumed to be IID gaussian with mean μ and variance σ^2. For incorporating the excess return as described by Sharpe (1994), the *ex-ante* Sharpe ratio is defined by

$$SR = \frac{\mu - r_f}{\sigma},$$

where r_f represents the risk-free rate. This equation incorporates a benchmark to the measure. As mentioned above, the mean μ and standard deviation σ has to be estimated. This can be done by the sample mean

$$\widehat{\mu} = \frac{1}{T}\sum_{i=1}^{T} r_t$$

and the sample variance

$$\widehat{\sigma^2} = \frac{1}{T}\sum_{i=1}^{T}(r_t - \widehat{\mu})^2.$$

This leads to the estimator of the Sharpe ratio

$$\widehat{SR} = \frac{\widehat{\mu} - r_f}{\widehat{\sigma}}.$$

To annualize this measure, $\widehat{SR}$ is typically multiplied by $\sqrt{a}$, which represents the average number of returns per year (Lo 2002). Since the true values of Sharpe ratio are unknown, its calculations may be subject of estimation errors (López De Prado 2018).

4.1.2 Maximum Drawdown

As stated by Hamelink & Hoesli (2004), the *maximum drawdown* can be described by considering a Price $P(t)$ and a overall maximum of all prices $P_{\max}(t)$

$$P_{\max}(t) = \max_{\tilde{t} \leq t} P(\tilde{t})$$

up to the time point t. Hence, the maximum drawdown (MaxDD) can be derived by

$$\text{MaxDD}(t^{\star}) = \max_{t \leq t^{\star}} \{P(t)/P_{\max}(t) - 1\}.$$

The MaxDD measures the stability of an asset or portfolio. It computes the loss, when the asset was bought at a local maximum and sold at the next local minimum. In general, this performance measure can be used to evaluate the risk of an investigated asset. In contrast to volatility, it provides a purly downside measuring value, whereas volatility is also affected by stock earnings (Hamelink & Hoesli 2004).

For this reason, MaxDD is introduced as additional measure to the backtest analysis. Figure 4.1 shows the DAX price series with the maximum drawdown marked as red line.

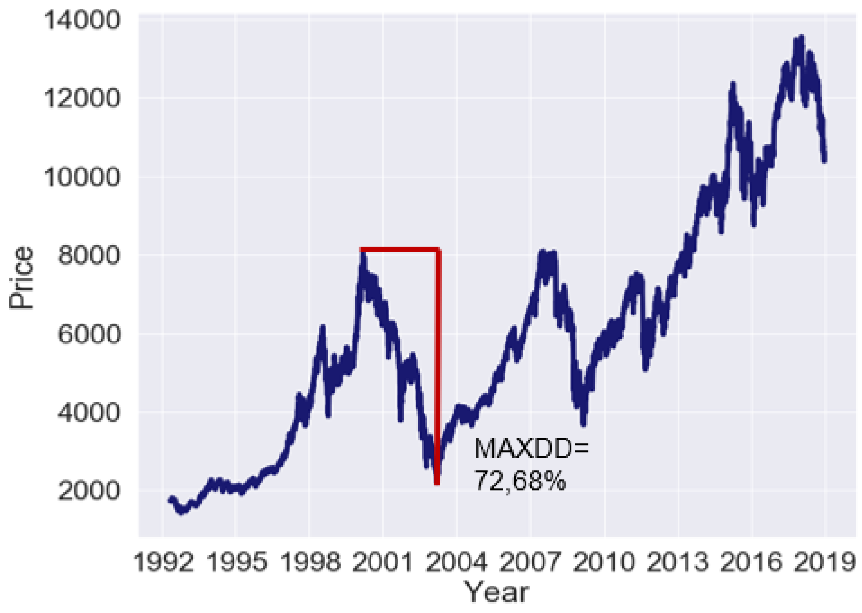


Figure 4.1: Maximum drawdown applied to DAX price series

4.1.3 Classification Scores

Classification scores are used to derive the performance of a machine learning algorithm. In general, it measures the ability of a model to identify labels by its associated features. For a better understanding of several classification scores, the confusion matrix is provided below:

		true value	
		pos	**neg**
predicted value	**pos′**	True positive	False positive
	neg′	False negative	True negative
	total	Pos	Neg

A good classifier has a high number of true positives and true negatives.

4.1.3.1 Area under the Receiver Operating Characterisitcs Curve

Another popular measure is the area under the receiver operating characteristics curve (ROC-curve). To compute the ROC-curve the *true positive rate*

$$\text{tp-rate} = \frac{\text{TP}}{\text{Pos}}$$

as well as the *false positive rate*

$$\text{fp-rate} = \frac{\text{FP}}{\text{Neg}}$$

are required. Then, the ROC-curve of a classifier is obtained by plotting the false positive rate against the true positive rate with different threshold values to classify the predictions. Hence, a higher area under the curve (AUC) suggests a better classifier. Computing the classifier by the tp-rate and the fp-rate penalizes classifier which simply predicts one class for each feature row. The ROC-AUC measure provides per definition a value within the range $[0.5, 1]$. Thus, a classifier which may have a high accuracy by predicting always the majority class will have a ROC-AUC of 0.5, whereas the perfect classifier receives a value of 1 (Fawcett 2006).

4.1.3.2 F1 Measure

As described by Zhou (2012), F1 is computed by

$$F1 = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2} \right)^{-1}.$$

F1 measure calculates the harmonic mean of precision or recall. Precision is defined by

$$\text{Precision} = \frac{TP}{TP + FP},$$

and measures how many classified positives are really postitive. It is equivalent to the true-postive rate defined above. Furthermore, recall is computed by

$$\text{Recall} = \frac{TP}{TP + FN}.$$

Recall evaluates the classifier prediction by computing the fraction of really positives towards the positive classified. For evaluating the models in the empirical analysis both performance measures, ROC-AUC and F1 are computed to capture the capacity of the models (Zhou 2012). Then the classifier's quality is interpreted by considering the four possible outcomes:

1. High ROC-AUC and high F1: The classifier performs very good independently of the selected threshold.
2. High ROC-AUC and low F1: The model produces overall bad results. However, by setting an appropriate threshold, the model can achieve a decent score.
3. Low ROC-AUC and high F1: The classifier currently delivers good results, though for many other threshold values, the classifier performs badly.
4. Low ROC-AUC and low F1: The model delivers poor results and even changing the threshold does not lead to better results.

4.2 Walk Forward Method

The most common backtest methodology in literature is the walk forward (WF) approach. A main characteristic of WF is that each strategy decision is based on observations that predates the decision. Typically, the dataset is divided into a training and hold out set, where observations of the hold out set join the training set after prediction. There are two main advantages of WF:

- WF shows exactly how a certain strategy would have behaved in the past, especially in the more turbulent market phases. Hence, WF has a clear historical interpretation.
- As stated by López De Prado (2018), using a WF window guarantees an out-of-sample testing set (no leakage), if the dataset gets purged properly.

WF is an intuitive approach. However, the main disadvantage of WF are listed below:

- As stated by Bailey and López de Prado (2014), only a single scenario is tested which can be easily overfitted.
- WF is not necessarily representative for future performance, results can be biased by the sequence of datapoints.
- Initial decisions are made on a smaller portion of the total sample. However, it is not valid for a rolling window, but a rolling window reduces the number of training observations used for predicting the test sample. Considering a strategy with a warm-up period leads to a shorter backtest path. Moreover, the strategy makes half of its decisions on an average amount of datapoints.

As can be seen, although WF has a clear historical interpretation, the approach has several disadvantages in particular providing only **one** backtest path (López De Prado 2018). Hence, only one Sharpe ratio, maximum drawdown, F1 and ROC-AUC can be computed. The impacts of these characteristics are clarified after describing the combinatorial purged cross-validation method.

4.3 Combinatorial Purged Cross-Validation Method

In this section a new backtest method namely the *combinatorial purged cross-validation* method (CPCV) is explained. Moreover, its differences to the walk forward method and the k-fold cross-validation are examined. In the CPCV approach the user predefines the expected number of backtest paths φ. Furthermore, a certain number of combinations of training/test sets have to be generated to build these paths, while purging training and embargoing training sets.

In the first step of the CPCV method, a certain number of splits are generated. For this approach T observations are partitioned into N groups without shuffling. There are $N-1$ groups of size $\lfloor T/N \rfloor$, whereas the Nth group is of size $T - \lfloor T/N \rfloor (N-1)$ and $\lfloor . \rfloor$ represents the floor function. Determing k as the number of testing sets out of N, the number of possible training/testing splits is given by

$$\binom{N}{N-k} = \frac{\prod_{i=0}^{k-1}(N-i)}{k!}.$$

For instance, a possible choice of $N = 5$ and $k = 2$ results in $\binom{5}{3} = 10$ different number of splits. The different combinations indexed as S1, ..., S10 are illustrated through the Table 4.1. Test partitions are marked with a cross, whereas train parts are not marked.

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	
G1	×	×	×	×							4
G2	×				×	×	×				4
G3		×			×			×	×		4
G4			×			×		×		×	4
G5				×			×		×	×	4

Table 4.1: Generated paths for $\varphi(5, 2) = 4$

The total number of test partitions is calculated as $k\binom{N}{N-k}$. Since each group N belongs to the same number of training and test sets, a total number of paths $\varphi(N, k)$ can be backtested:

$$\varphi(N, k) = \frac{k}{N}\binom{N}{N-k} = \frac{\prod_{i=1}^{k-1}(N-i)}{(k-1)!}.$$

The assignment of each tested group of the previously described example is shown in Table 4.2

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	
G1	**1**	**2**	**3**	**4**							4
G2	**1**				**2**	**3**	**4**				4
G3		**1**			**2**			**3**	**4**		4
G4			**1**			**2**		**3**		**4**	4
G5				**1**			**2**		**3**	**4**	4

Table 4.2: Generated paths for $\varphi(5, 2) = 4$, Assignment of testing groups

Hence, the first backtest path is built of the predictions from (G1, S1), (G2, S1), (G3, S2), (G4, S3) and (G5, S4), whereas the second is built from (G1, S2), (G2, S5), (G3, S5), (G4, S6) and (G5, S7), and so on. In this example a total number of four backtest paths are generated, by training a classifier on a portion $\iota = 1 - k/N$ of the data for each combination. Even if it is possible to create portions $\iota < \frac{1}{2}$, it is not recommended to consider a higher fraction of test data towards training data. Therefore, in practice $k \leq N/2$ is assumed. This assumption leads to several characteristics of the CPCV approach. Thereby, the portion of data considered in the training set ι increases with $N \to T$, but decreases with $k \to N/2$, whereas the number of paths $\varphi(N, k)$ increases with $N \to T$ and with $k \to N/2$. Hence, the number of paths can be maximized by setting $N = T$ and $k = N/2 = T/2$. However, this can only be executed at the expense of training the classifiers on only half of the data for each combination $\iota = 1/2$ (López De Prado 2018).

As stated by López De Prado (2018), an easy rule of thumb is to determine $k = 2$ in order to partition the data into $N = \varphi + 1$ groups, where φ is the number of targeted paths. In general, the case $k = 2$ will generate $\varphi(N, 2) = N - 1$ paths, which is particularly interesting because the classifier will train on a large portion of data $\iota = 1 - 2/N$ and almost as many backtest paths as the number of groups $N - 1$ are generated. In the limit $N = T$ one observation could be assigned to one group and $\varphi(T, 2)$ could be created. Then the classifier is trained on a portion of $\iota = 1 - 2/T$ of the data per combination.

According to López De Prado (2018), the combinatorial purged cross-validation can be summarized as described in Algorithm 3:

Algorithm 3 The Combinatorial Purged Cross-Validation Backtest Algorithm

```
1. Partition $T$ observations into $N$ groups without shuffling such, that $N-1$ groups are of size $\lfloor T/N \rfloor$, whereas the $N$th group is of size $T - \lfloor T/N \rfloor (N-1)$ and $\lfloor . \rfloor$ represents the floor function.

2. Compute all possible training/test splits, where for each split $N-k$ groups constitute training set and $k$ groups constitute the testing set.

3. Apply purging to any pair of labels $(y_i, y_j)$, where $y_i$ belongs to the training set and $y_j$ belongs to the test set. An embargo should be considered if testing samples predate some training samples.

4. Fit a pre-defined classifier on the $(N-k)\binom{N}{N-k}$ training sets and produce forecasts on the respective $k\binom{N}{N-k}$ test sets.

5. Compute the $\varphi(N,k)$ backtest paths. Calculate the Sharpe ratio or any other backtest measure like maximal drawdown, area under the curve and F1, from each path and derive the empirical distribution of each backtest measure.
```

To examine the advantages of the CPCV algorithm over walking forward (WF) or k-fold CV, the variance of both approaches are investigated. Considering a set of $\mathcal{S}$ independent backtests associated with a certain strategy class. These elements of the set are called *trials* and a Sharpe ratio $\widehat{\mathrm{SR}}_s$ with $s = 1, \dots, \mathcal{S}$ can be estimated on every trial. To compute the expected maximum $\mathbb{E}[\max\{\widehat{\mathrm{SR}}_s\}_{s=1,\dots,\mathcal{S}}]$ of these Sharpe ratios, an investigation of the expected maximum of IID random variables $x_s \sim \Phi, s = 1, \dots, \mathcal{S}$ is helpful. When Φ represents the standard normal distribution the expected maximum of x_s can be approximated as

$$\mathbb{E}[\max\{x_s\}_{s=1,\dots,\mathcal{S}}] \approx (1-\gamma)\Phi^{-1}\left[1 - \frac{1}{\mathcal{S}}\right] + \gamma\Phi^{-1}\left[1 - \frac{1}{\mathcal{S}}e^{-1}\right] \leq \sqrt{2\log(\mathcal{S})},$$

where $\gamma \approx 0.5772$ is the Euler-Mascheroni constant and $\mathcal{S} >> 1$ (compare proof in A.1). In addition, the inverse of the CDF of Φ represents $\Phi[.]^{-1}$ and the derivation of the upper bound $\sqrt{2\log(\mathcal{S})}$ is described by Embrechts *et al.* (1997). Hence, with increasing $\mathcal{S}$, it is expected to find a higher expected maximum $\mathbb{E}[\max\{x_s\}_{s=1,\dots,\mathcal{S}}]$.

Furthermore, it is assumed that the underlying instrument behaves like a martingale. The martingale exposes $\mathbb{E}[\{\widehat{\mathrm{SR}}_s\}_{s=1,\ldots,\mathcal{S}}] = 0$ and $\mathrm{Var}[\widehat{\mathrm{SR}}_s] > 0$, whereas $\frac{\widehat{\mathrm{SR}}_s}{\mathrm{Var}[\widehat{\mathrm{SR}}_s]} \sim \Phi$ is supposed. Hence, the expected maximum Sharpe ratio can be derived by

$$\mathbb{E}[\max\{\widehat{\mathrm{SR}}_s\}_{s=1,\ldots,\mathcal{S}}] = \mathbb{E}[\max\{x_i\}_{i=1,\ldots,I}]\sqrt{\mathrm{Var}[\widehat{\mathrm{SR}}_s]}.$$

Although the true Sharpe ratio is zero, it is expected to find one trial with a Sharpe ratio equal to the just noted equation (Bailey & Lopez de Prado 2014). This equation is crucial for understanding. As stated by López De Prado (2018), researcher will select the backtest with the maximum *estimated* Sharpe ratio, even if the true Sharpe ratio is zero. Hence, it is necessary to note the number of trials $\mathcal{S}$ in walk-forward backtesting. The Sharpe ratio calculated out of WF exhibits a high variance $\mathrm{Var}[\widehat{\mathrm{SR}}_s]$ because a large portion of the decisions are based on a small portion of the dataset. This can be reduced by including a warm-up period, but this will reduce the length of the backtest and therefore increases the variance (López De Prado 2018).

In contrast to WF, CV backtest exhibits a lower variance, since each classifier is trained on an equal portion of the whole dataset. Moreover, the backtest path is longer which reduces the variance. However, WF and CV still estimate Sharpe ratio from a single path for a trial. Thus, the estimate $\widehat{\mathrm{SR}}_s$ may be high volatile. As opposed to WF and CV, CPCV derives its Sharpe ratios from a large number of paths, with mean $\mathbb{E}[\widehat{\{\mathrm{SR}_{s,j}}\}_{j=1,\ldots,\varphi}] = \mu_s$ and variance $\mathrm{Var}[\widehat{\mathrm{SR}}_s] = \sigma_s^2$. Therefore, the variance of the CPCV paths sample mean can be calculated by

$$\mathrm{Var}[\mu_s] = \varphi^{-2}(\varphi\sigma_s^2 + \varphi(\varphi-1)\sigma_s^2\overline{\rho}_s) = \varphi^{-1}\sigma_s^2(1 + (\varphi-1)\overline{\rho}_s).$$

Referring to this equation, $\overline{\rho}_s$ represents the average off-diagonal correlation among $\widehat{\{\mathrm{SR}_{s,j}}\}_{j=1,\ldots,\varphi}$, whereas σ_s^2 serves as variance of the Sharpe ratios across paths for strategy s. In general, CPCV has the same goal as bootstrapping: Reducing variance. Hence, $\overline{\rho}_s < 1$ implies that the variance of the sample mean is lower than the variance of the sample

$$\frac{\sigma_s^2}{\varphi} \leq \mathrm{Var}[\mu_s] < \sigma_s^2.$$

A decreasing correlation between the paths leads to a lower CPCV variance. In the limit CPCV will report the true Sharpe ratio $\mathbb{E}[\{\widehat{\mathrm{SR}_s}\}] = 0$ with zero variance because $\lim_{\varphi\to\infty}\mathrm{Var}[\mu_s] = 0$. However, this cannot be achieved, since φ has an upper bound $\varphi \leq \varphi[T, \frac{T}{2}]$. In general, the high number of paths leads to a more confident estimate of the true Sharpe ratio and leads to fewer false discoveries in contrast to WF and CV (López De Prado 2018).

5 Empirical Analysis

5.1 Descriptive Analysis

The following section provides a descriptive overview of the DAX and the REXP, as well as the return structure of the DAX series after applying the CUSUM filter on it. The DAX index contains the 30 biggest and most liquid companies of the German stock market, whereas the REXP is a performance index for German government bonds. For the data analysis the time period between April 1992 and December 2018 was investigated and daily data is used to perform the data analysis. Hence, the dot-com crash 2000-2002 and the subprime crisis 2007-2008 are included in the investigated dataset. The dot-com crash, for instance, causes the DAX to lose almost all of its earnings of 1992. However, the REXP exhibits a nearly linear trend from 1992 to 2018. This is visible in Figure 5.1 (plotted on Log-scale).

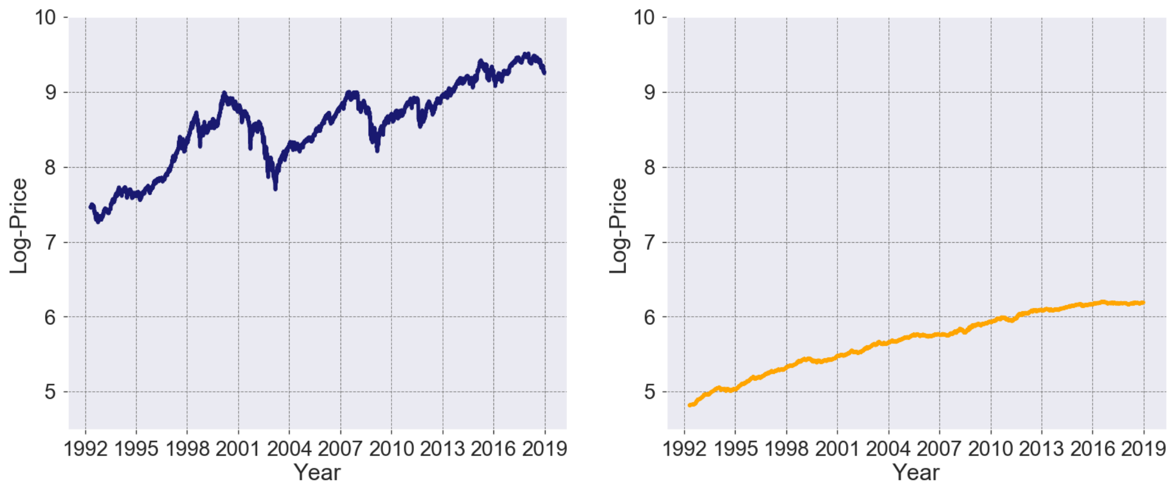


Figure 5.1: Price series DAX/REXP

Overall, the REXP seems more stable, especially in the financial crisis. This can be also investigated in the log-returns of the time series, displayed in Figure 5.2, which is based on a sample size of $T = 6753$ observations. Furthermore, typical characteristics like volatility cluster and other stylized facts can be observed. Table 5.1 confirms this statement with some relevant statistics of percentage log-returns. Both series have a positive mean, whereas the DAX log-returns has a far higher standard deviation than the REXP log-returns. The REXP log-returns are left skewed while the DAX log-returns are slightly right skewed. The kurtosis conveys that both series have a sharp peak around the mean and especially the DAX log-return series has heavy tails.

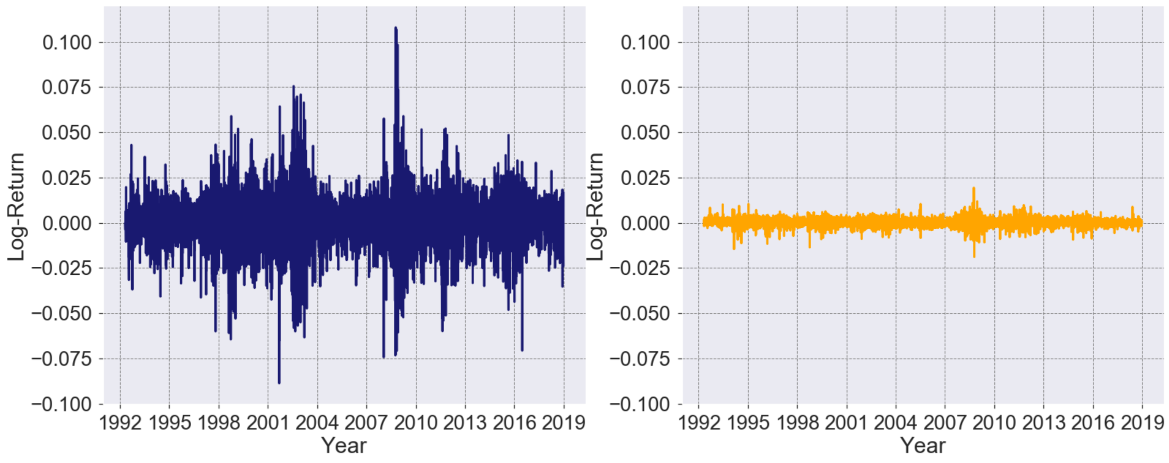


Figure 5.2: Log-return series DAX/REXP

Index	Median	Mean	Std.Dev.	Min	Max	Skew	Kurt
DAX	0.079	0.027	1.408	-8.875	10.798	11.83	447.66
REXP	0.024	0.020	0.214	-1.894	1.935	27.40	524.81

Table 5.1: Summary of DAX log-returns

In Figure 5.3 the DAX log-return series is plotted against a gaussian distribution. The compared distribution is computed by gaussian random variables with the same mean and variance as the DAX log-returns. As mentioned previously, both time series (REXP distribution visible in Figure A.2) have heavy tails and have a sharp peak around their mean. In addition, the correlation between the two time series is -0.105.

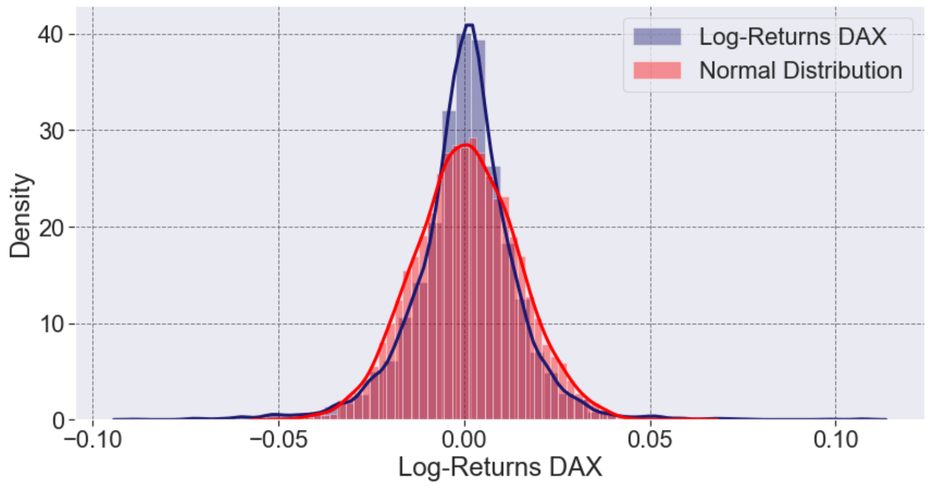


Figure 5.3: Histogram DAX

As stated by López De Prado (2018), a machine learning algorithm wants to learn from relevant examples. As explained in subsection 2.2, the symmetric CUSUM filter is a helpful tool to achieve this objective. By setting the parameter h to two times the standard deviation of the log-returns $h = 2 \times \sigma_{\text{DAX Log-returns}} = 0.028$, the symmetric CUSUM filter is applied to the DAX time series. Figure 5.4 illustrates the resulting distribution.

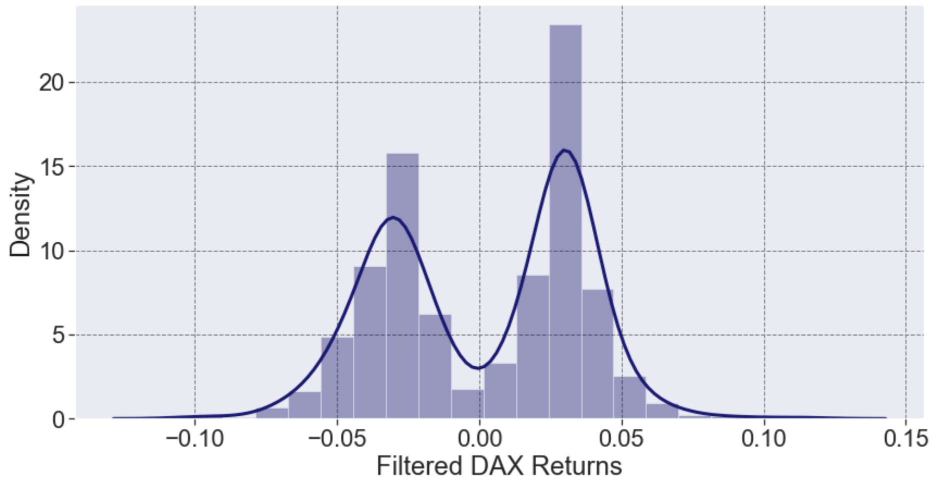


Figure 5.4: Filtered DAX

The original unimodal distribution changes to a bimodal distribution. This result is expected since applying CUSUM filter deletes values under the threshold h. As mentioned in subsection 2.2, a big disadvantage of this method is the rather arbitrary choice of the threshold h.

Before the modelling part, the labels must be computed. For this purpose, the triple-barrier method, described in subsection 2.3.1, is applied to the filtered dataset. As a vertical barrier 60 days equivalent to ≈ 42 trading days are selected for the expiration limit. Since potential losses and gains are equally weighted, the horizontal barriers are constructed to be symmetric. These bandwidths depend on the estimated volatility. As stated in subsection 2.3.1, the volatility is estimated by an exponential moving average method with a span of 20. To get a binary classification problem, a hit at the vertical barrier is labelled as the sign of the return. Overall, this procedure yields the number of class labels shown in Table 5.2.

	upp	**low**	**vert**
# Hits	507	351	272

$\Rightarrow$

Positive Label	**Negative Label**
645	485

Table 5.2: Barrier hits converted to labels

Hence, positive labels form the majority of the observations with a share of 57.1%. Another important step to improve modelling is applying sample weights to the observations. As described in subsection 2.4, the sample weights are computed by using their absolute amount and their uniqueness. Consequently, for bagging and random forest more indpendent bootstrap samples are considered. For the filtered time series used in further applications, the computation of sample weights yields a positive skewed distribution (cf. Figure 5.5). The median weight equals to 0.86, whereas the mean value equals to 1. Furthermore, the largest weights represent enormous turbulent dates with respect to the financial markets.

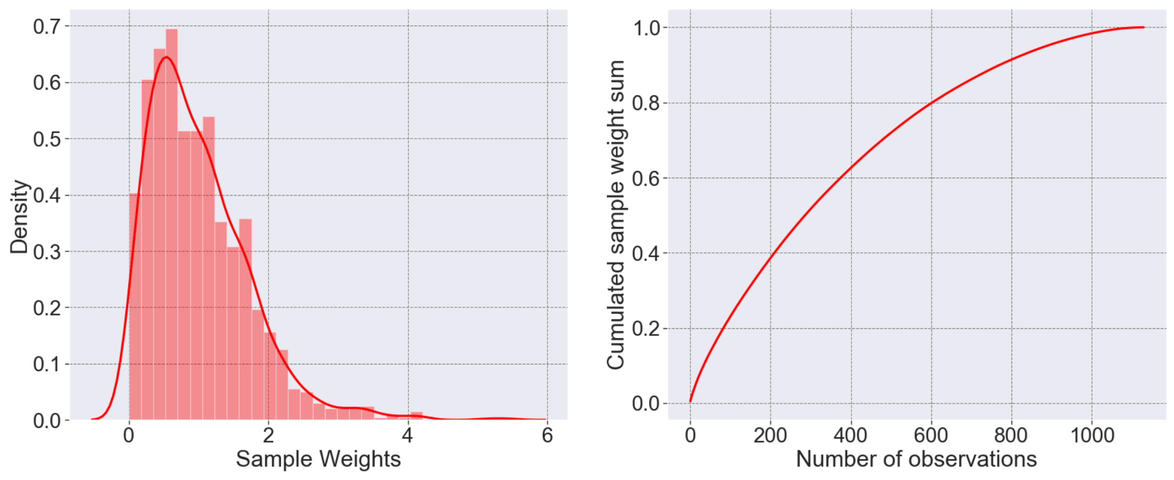


Figure 5.5: Sample weight distribution

The maximum weighted observation is represented by the 11 September 2001. This day represents the loss in the DAX caused by the terrorist attacks in New York. In addition, this observation is also the most negative return in the series. Thus, this observation is weighted with a value of 5.4, which is about 5 times higher than an average weighted value in the sample weight vector. In the right graphic of Figure 5.5 the sample weights are sorted by size and their cumulative sum is shown. As illustrated in the figure, around 35% of the data represents about 60% of the overall sample weights. Comparing this with equally weighted observations leads to the conclusion that introducing sample weights can lead to a heavy impact in terms of modelling.

5.2 Selected Features

In the following content the engineered features for the data analysis are described and their correlation is computed. All features are built from the DAX price series. Furthermore, a Principal Component Analysis (PCA) was performed on the data. One possible feature is the fractional differentiated price series. As explained in subsection 2.1, this approach can lead to a good feature for predicting financial observations. To utilize this approach several values between 0 and 1 are considered for the fractionalization parameter d. Then the Fixed-Width Window Fracdiff method is used to compute the fractional differentiated series. This procedure is printed in Table 5.3.

d	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
ADF stat	-1.70	-1.83	-2.02	-2.29	-2.59	-2.99	-3.39	-4.02	-4.62
p-value	0.43	0.36	0.28	0.18	0.09	0.04	0.01	0.00	0.00

Table 5.3: ADF statistics results for fractional differentiation of DAX time series

For considering $\alpha = 0.05$, the value $d = 0.35$ is suggested, which is illustrated in A.3. Supposing this value for fractional differentiating leads to the time series displayed in Figure 5.6.

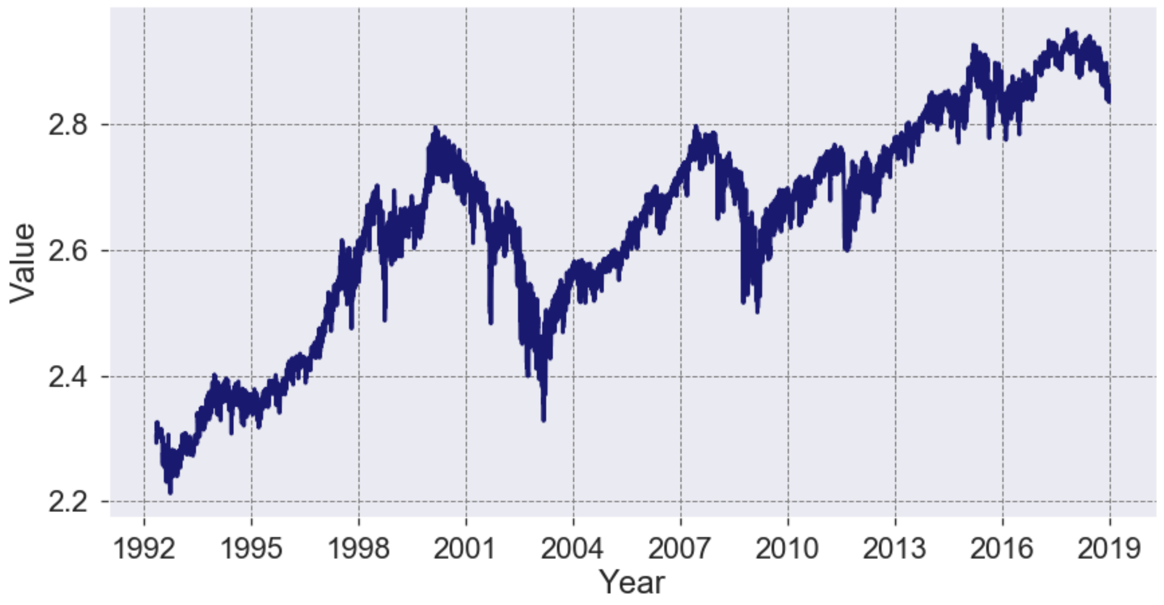
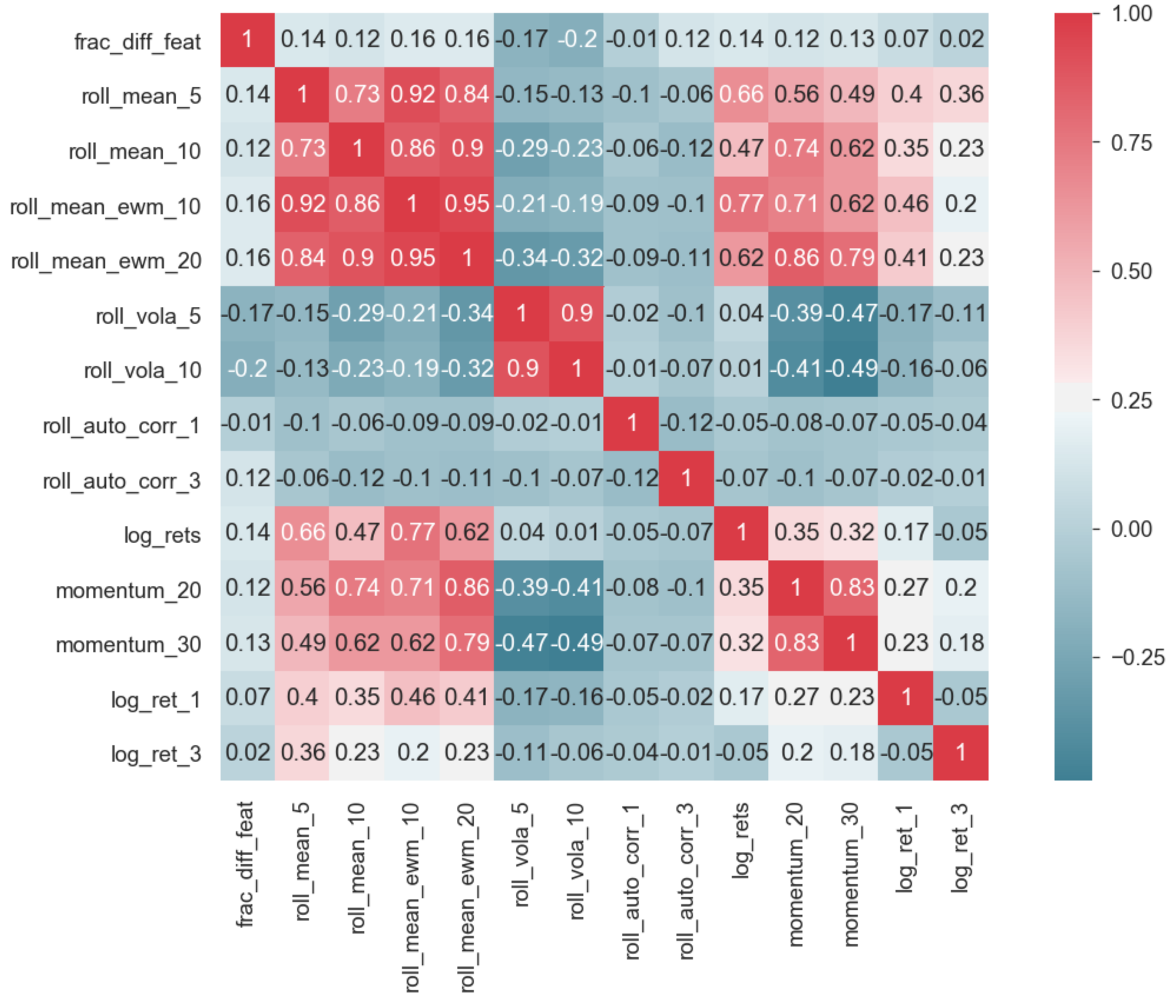


Figure 5.6: Fractional differentiated series

Overall, the mean of the time series in Figure 5.6 varies over time and consequently weak stationarity is questionable. The fractionalized differentiated series is only one feature of many used in this thesis. A list of features engineered for the analysis is presented below. These are similar to those selected by Bao *et al.* (2017). Instead of the filtered dataset all features are computed using the original DAX price series or log-returns. Then the filtered events are considered for the final feature matrix.

1. Rolling mean
    - 5-day Moving average (roll_mean_5)
    - 10-day Moving average (roll_mean_10)
2. Rolling exponential weighted mean (EWM)
    - 10-day EWM (roll_mean_ewm_10)
    - 20-day EWM (roll_mean_ewm_20)
3. Rolling volatility
    - 5-day Moving standard deviation (roll_vola_5)
    - 10-day Moving standard deviation (roll_vola_10)
4. Momentum feature
    - Percent price change of four business weeks (momentum_20)
    - Percent price change of six business weeks (momentum_30)
5. Autocorrelation
    - 20-day autocorrelation (lag = 1) (roll_auto_corr_1)
    - 20-day autocorrelation (lag = 3) (roll_auto_corr_3)
6. Differentiation features
    - Log-Returns (log_rets)
    - Log-Return shift1 (log_rets_1)
    - Log-Return shift3 (log_rets_3)
    - Fracd series (frac_diff_feat)

To receive information about the correlations between the seperate features, the spearman correlation coefficient is computed between each pair of feature. Figure 5.7 indicates strong correlation between the moving average features and the momentum feaures. High correlations can be a problem in modelling in terms of multicollinearity. In the case of perfect multicollinearity, the moment matrix has not full rank. Hence, the inverse of the moment matrix can not be computed. As explained in subsection 3.2, the logistic regression alleviates this problem by imposing a size constrain on the coefficients. Furthermore, applying principal component analysis to the dataset solves this issue.



	frac_diff_feat	roll_mean_5	roll_mean_10	roll_mean_ewm_10	roll_mean_ewm_20	roll_vola_5	roll_vola_10	roll_auto_corr_1	roll_auto_corr_3	log_rets	momentum_20	momentum_30	log_ret_1	log_ret_3
frac_diff_feat	1	0.14	0.12	0.16	0.16	-0.17	-0.2	-0.01	0.12	0.14	0.12	0.13	0.07	0.02
roll_mean_5	0.14	1	0.73	0.92	0.84	-0.15	-0.13	-0.1	-0.06	0.66	0.56	0.49	0.4	0.36
roll_mean_10	0.12	0.73	1	0.86	0.9	-0.29	-0.23	-0.06	-0.12	0.47	0.74	0.62	0.35	0.23
roll_mean_ewm_10	0.16	0.92	0.86	1	0.95	-0.21	-0.19	-0.09	-0.1	0.77	0.71	0.62	0.46	0.2
roll_mean_ewm_20	0.16	0.84	0.9	0.95	1	-0.34	-0.32	-0.09	-0.11	0.62	0.86	0.79	0.41	0.23
roll_vola_5	-0.17	-0.15	-0.29	-0.21	-0.34	1	0.9	-0.02	-0.1	0.04	-0.39	-0.47	-0.17	-0.11
roll_vola_10	-0.2	-0.13	-0.23	-0.19	-0.32	0.9	1	-0.01	-0.07	0.01	-0.41	-0.49	-0.16	-0.06
roll_auto_corr_1	-0.01	-0.1	-0.06	-0.09	-0.09	-0.02	-0.01	1	-0.12	-0.05	-0.08	-0.07	-0.05	-0.04
roll_auto_corr_3	0.12	-0.06	-0.12	-0.1	-0.11	-0.1	-0.07	-0.12	1	-0.07	-0.1	-0.07	-0.02	-0.01
log_rets	0.14	0.66	0.47	0.77	0.62	0.04	0.01	-0.05	-0.07	1	0.35	0.32	0.17	-0.05
momentum_20	0.12	0.56	0.74	0.71	0.86	-0.39	-0.41	-0.08	-0.1	0.35	1	0.83	0.27	0.2
momentum_30	0.13	0.49	0.62	0.62	0.79	-0.47	-0.49	-0.07	-0.07	0.32	0.83	1	0.23	0.18
log_ret_1	0.07	0.4	0.35	0.46	0.41	-0.17	-0.16	-0.05	-0.02	0.17	0.27	0.23	1	-0.05
log_ret_3	0.02	0.36	0.23	0.2	0.23	-0.11	-0.06	-0.04	-0.01	-0.05	0.2	0.18	-0.05	1

Figure 5.7: Correlation matrix

Before choosing a model for these features, the feature importance is measured by the three methods described in subsection 3.5.2 with a random forest. As stated by López

De Prado (2018): "Backtesting while researching is like Drinking and Driving. Do not research under the influence of a backtest." Therefore, before a certain strategy is selected, research must be finished. Since every method has advantages and disadvantages, all three methods are considered to measure feature importance. Furthermore, every method will be performed with and without sample weights in order to conduct backtesting afterwards.

5.2.1 MDI

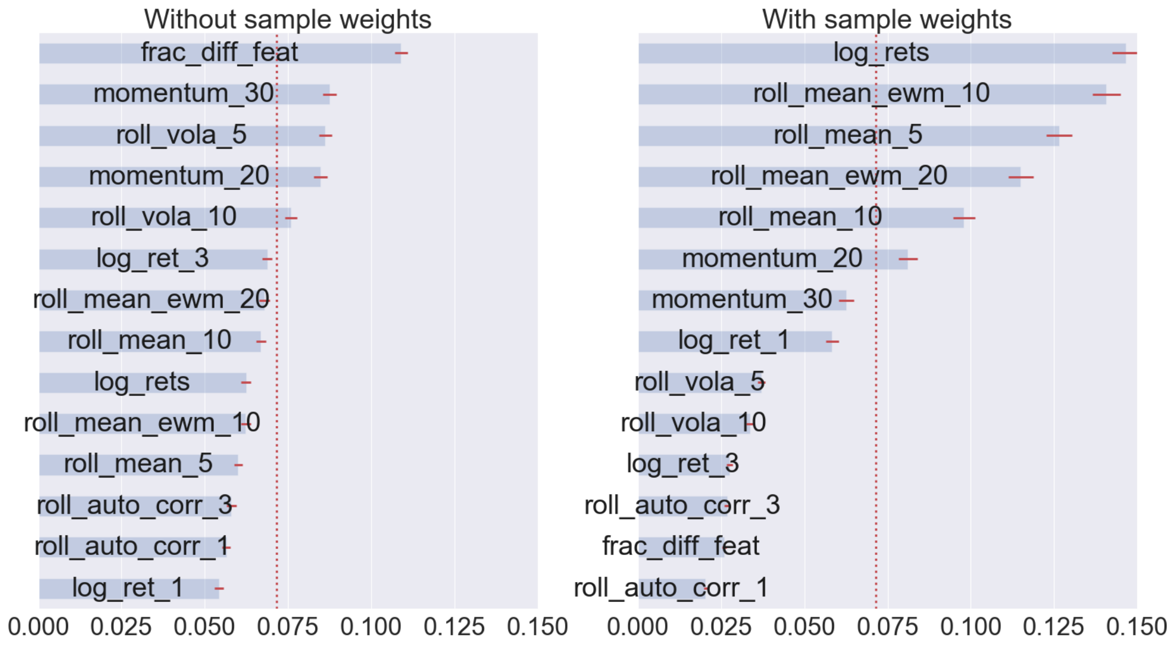


Figure 5.8: MDI feature importance

As stated in subsection 3.5.2, feature importance measured by mean decrease impurity (MDI) is based on the entropy in the potential split points. By computing several trees with bootstrapped samples, a certain value can be assigned to each feature by counting the number of split points in the trees. In Figure 5.8 the feature importance based on MDI is plotted with (right graphic) and without (left graphic) sample weights. The horizontal lines are the standard deviations of the means, whereas the dotted vertical line represents the feature importance value if all features are weighted equally, so 1/14. Hence, this line separates features whose importance exceeds what would be expected from undistinguishable features. Without incorporating sample weights, the rolling volatility features, momentum features and the fractional differentiated DAX are ranked above the dotted red line. In contrast to this result, by incorporating sample weights, the log-returns and the rolling average features are identified as important, whereas rolling volatility and the fractional differentiated series are ranked very low.

5.2.2 MDA

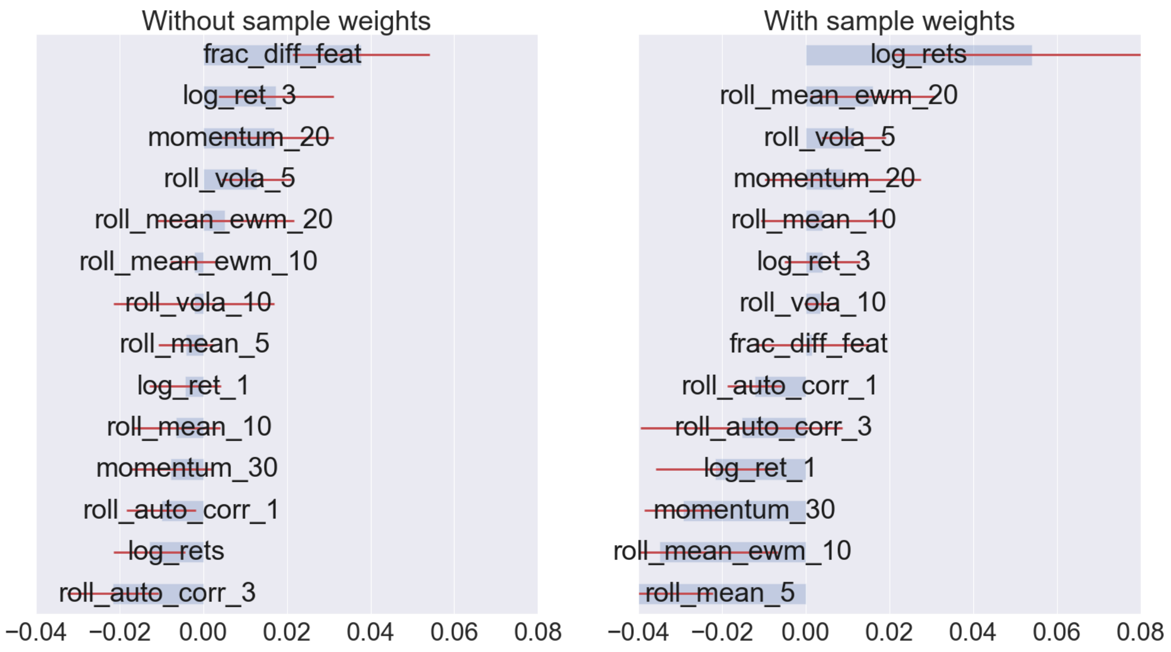


Figure 5.9: MDA feature importance

By using MDA as feature importance tool, the figure is different in terms of the x-axis. In Figure 5.9 the out of sample improvement of the feature regarding the performance measure is denoted by the x-axis. It measures the percentage change of the ROC-AUC measure when the repective feature is not permuted in contrast to the permuted case. For instance, the log_rets feature leads to an average improvement of 5.5% in contrast to the case when log_rets is permuted in the case with sample weights. It is also visible that all ranked features have a high standard deviation. The best features considering no sample weights are the fracdiff series, log return change of the last three trading days, the percent price change of two business weeks and the 5-day moving standard deviation. Incorporating sample weights delivers again different results. Log-return is ranked as the best predictive feature and the 20-day EWM as second best. In both applications the 5-day rolling standard deviation and the percent price change of two business weeks are ranked relatively high. Furthermore, all features in both approaches crosses the null value on the x-axis except for the three best ranked features respectively.

5.2.3 SFI

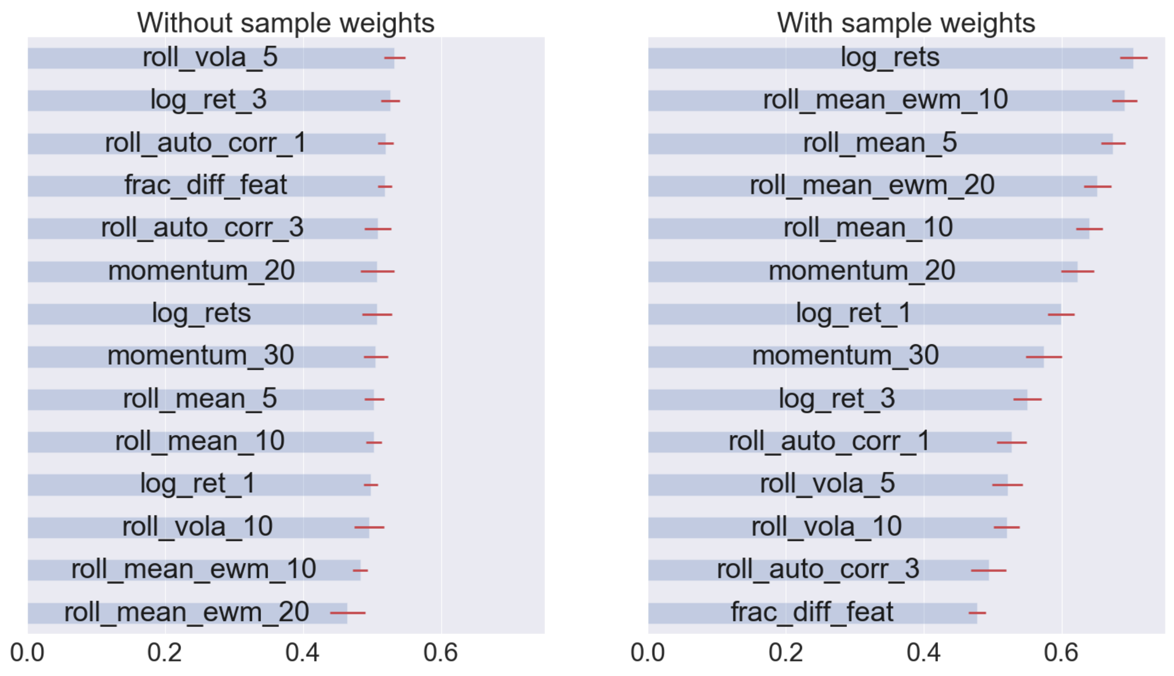


Figure 5.10: SFI feature importance

Figure 5.10 exhibits the respective feature importances resulting from the single feature importance approach. Single feature importance measures the influence of a feature independent of the others. Without incorporating sample weights, all features reach with a ROC-AUC around 0.5 rather similar results. This means that a feature alone has no predictive power. Considering sample weights leads to different results. Log-returns are ranked again as the best feature. Other high ranked features are the rolling mean features. They reach a better score than without sample weights. Only considering Figure 5.10 would lead to the conclusion that incorporating sample weights features have some predictive power. However, this holds not for the case without sample weights.

One simulation run will include PCA applied to the features. For this reason, the PCA is first applied to the filtered data. As described by Alexander (2001), PCA reduces the dimension of the feature matrix X by orthogonalizing the variables. Thus, the feature matrix gets diagonal. PCA has several advantages. It is an efficient way to deal with the multicollinearity problem. After applying PCA, efficient parameter estimation is possible, even if the feature matrix implies strong correlated features. Since there are strong estimated correlations between the rolling mean features, PCA might be a good transformation for receiving better model outcomes. As stated by López De Prado (2018), re-scaling the feature matrix X is an important step before applying PCA. The first principal component is dominated by the high variance of some features without re-scaling. No deeper relationship between features would be revealed. Thus, the feature matrix gets standardized before PCA. As an illustration, PCA is applied on the filtered feature matrix. Figure 5.11 shows the cumulative sum of the explained variance per number of components.

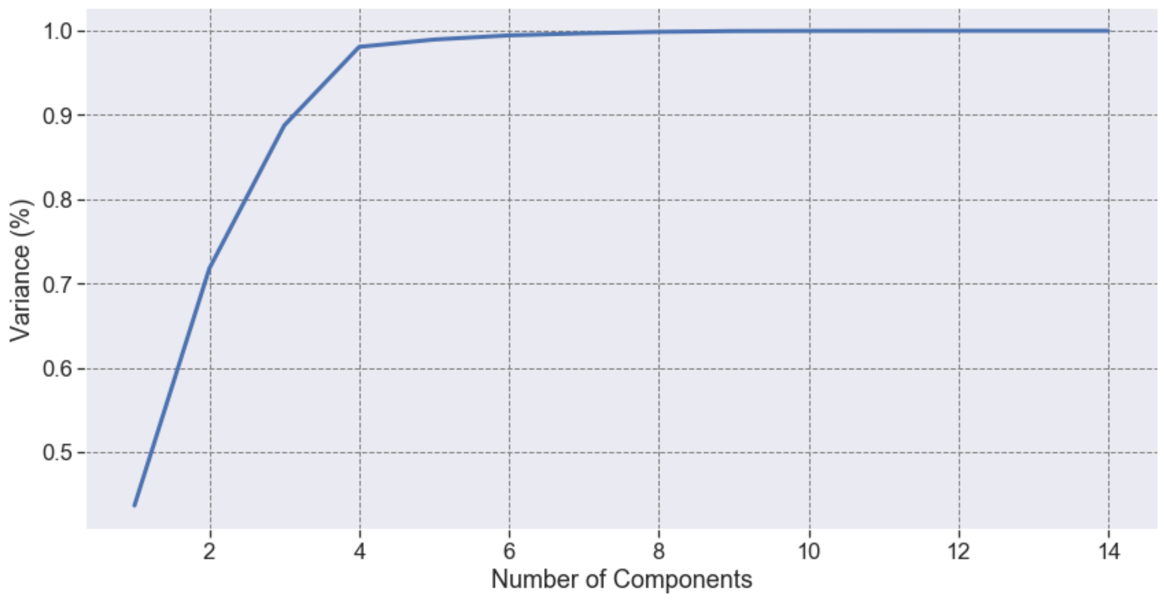


Figure 5.11: Explained variance, PCA

In the simulations the principal components are selected such that 95% of the variance is explained. Figure 5.11 shows that four principal components are enough to reach this target.

5.3 Backtesting Results

In the following section the backtesting results of four different models

- Logistic Regression (LogR)
- Bagged Logistic Regression (B_LogR)
- Bagged Decision Trees (B_Trees)
- Random Forest (RF)

are considered with different base number of features introduced in the former section. Overall, the models are applied to the filtered data with all features, the respective principal components, and a determined collection based on the results of the feature importance plots. The models are evaluated based on performance measures ROC-AUC, F1, Sharpe ratio and maximum drawdown. Since the CPCV algorithm is used for backtesting, instead of a single result a distribution of results is provided. In this thesis $\varphi = 30$ backtest paths are generated. Therefore, considering the rule of thumb given by López De Prado (2018), determining $k = 2$ results to $N = \varphi + 1 = 31$ partitions. Sampling two partitions out of 31 without replacement leads to overall 465 different combinations of training and test partitions. As stated in subsection 4.3, in each of this 465 combinations a pre-defined model is fitted on the 29 training partitions and predicts the observations of the other two partitions. Out of this predictions, 30 backtest paths are created.

For modelling a two-stage framework as explained in subsection 2.3.2 is employed. As stated by López De Prado (2018), the primary model tries to achieve a high recall. Thus, most positives are identified by the primary model. For this purpose, the primary model is tuned on the respective training data in order to achieve a high recall. After the side of the model has been determined by the primary model, meta-labelling is applied and the so far feasible labels transform from $\{-1, 1\}$ to $\{0, 1\}$. In addition, the side feature provided by the primary model is considered for further modelling. In the next step, the secondary model is tuned by the F1 score to filter out the false positives. The secondary model gives a suggestion whether to adjust the portfolio (1) or not (0). Adjusting the portfolio means that the secondary model suggests to acting or passing based on the given features and the side prediction from the primary model. The overall procedure is roughly summarized in Figure 5.12.

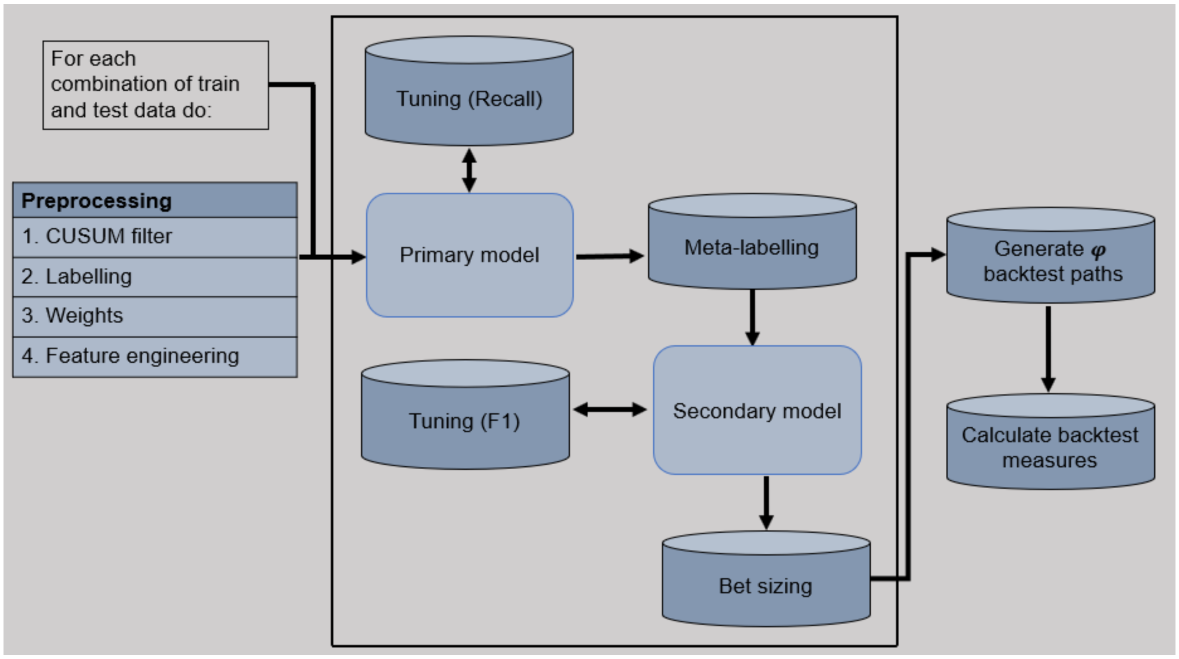


Figure 5.12: Model procedure

For tuning both the primary and the secondary model, one parameter set is provided respectively. Only the logistic regression and the decision tree is tuned. As suggested by López De Prado (2018) 1000 trees are considered for random forest and bagging. This ensures that the final majority vote is based on a lot of results provided by the respective decision trees.

Model	Logistic Regression	Decision Tree
Parameter	• Penalty: [L1, L2] • Regl. str. [0.01,0.1, 1]	• Tree depth: [3, 4, 5, 6] • Min.weight.frc. [0.05, 0.075, 0.1]

Table 5.4: Summary of tuning parameters

To prevent overfitting a rather strong regularization parameter set is provided. *Regl. str.* represents the regularization strength in the logistic regression which is useful in the case of collinear features and to prevent overfitting. These weights are evaluated with both L1 and L2 loss. Overfitting of decision trees is controlled by the maximum depth of the trees. A tree with many nodes is likely to overfit financial data. In addition, *Min.weight.frc.* represents the minimum weighted fraction of all input samples required in a leaf node. For instance, if the parameter equals to 0.05, at least 0.05% of the input samples must be in a leaf node, otherwise the leaf node is not created.

In the following, the investing strategy is briefly discussed. After receiving prediction probabilities of the model environment, these outcomes must be converted to bet sizes. The bet size is calculated as described in subsection 2.5. The computed bet sizes are used as portfolio weights for the DAX index. Since the DAX has a bigger maximum drawdown and a higher standard deviation as the REXP, only a very confident model should lead to higher portfolio weights. However, the DAX index has also a higher mean. For this reason, the DAX weight in the portfolio increases with the probability size of the model outcome. Considering the overall invested wealth Ψ_i at time point i, the following equation expresses this strategy

$$\Psi_i = b_i^{\mathrm{DAX}} \Psi_i + (1 - b_i^{\mathrm{DAX}}) \Psi_i = \Psi_i^{\mathrm{DAX}} + \Psi_i^{\mathrm{REXP}}$$

In the following subsections the models are backtested with all features, features selected by a PCA and a pre-defined selection of features. Sharpe ratio, maximum drawdown, ROC-AUC and F1 are computed on 30 backtest paths. The resulting performance distributions are visualized with boxplots. An single REXP portfolio has a yearly Sharpe ratio of 1.1 and a maximum drawdown of 0.05, whereas a 100% DAX portfolio has a maximum drawdown of 0.72 and a Sharpe ratio of 0.45. Short selling is not allowed and the respective negative bets are set to Null.

5.3.1 All Features

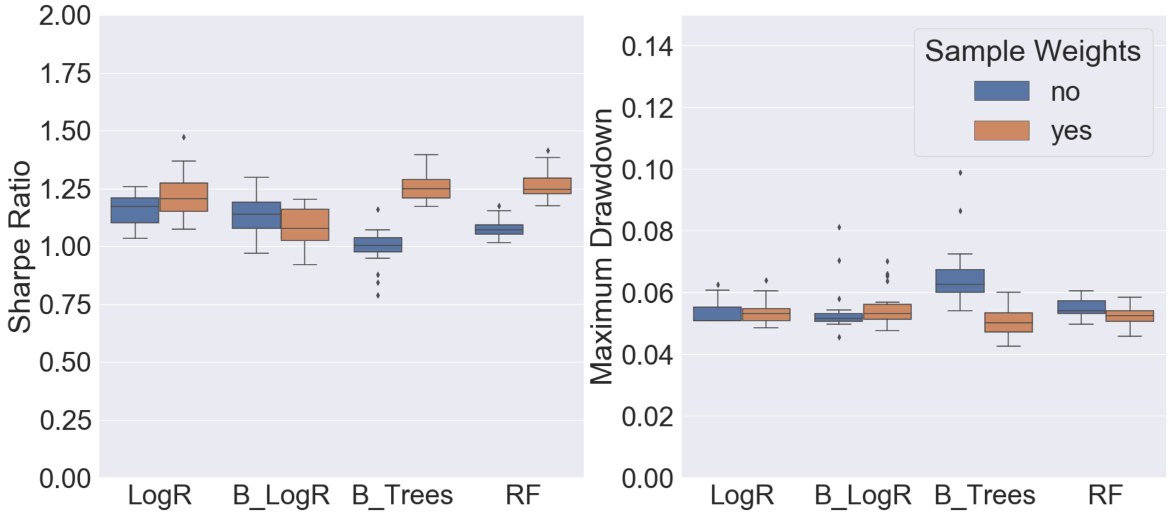


Figure 5.13: All features: Sharpe ratio, maximum drawdown

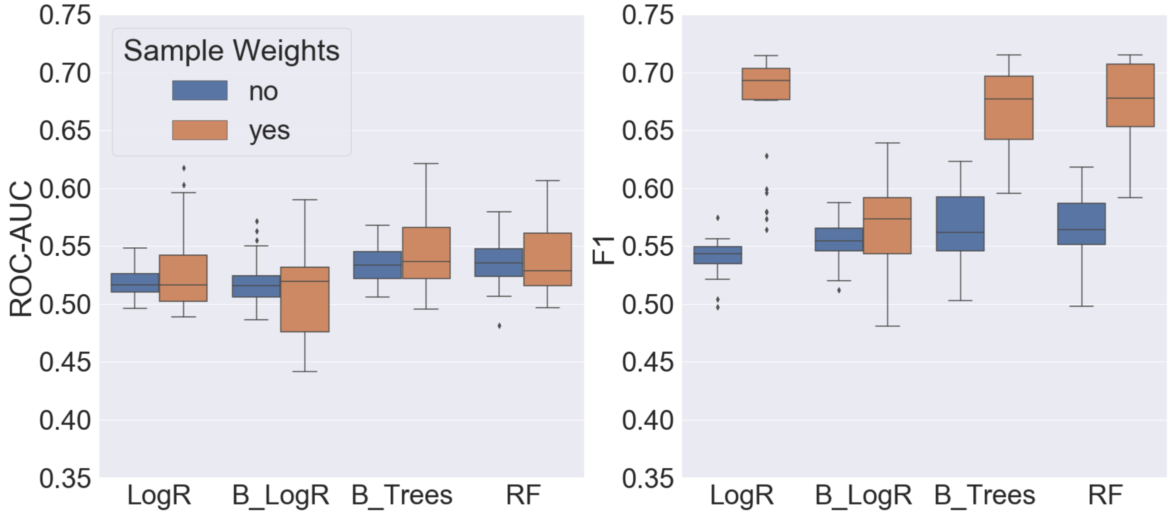


Figure 5.14: All features: ROC-AUC, F1

Figure 5.14 shows rather poor results for the application where all features are considered. According to the ROC-AUC, the classification power of all models is weak, though the F1 measure is larger, when applying sample weights. The logistic regression has a high median F1 measure. However, the median of the ROC-AUC is near 0.5. When applying sample weights, logistic regression, random forest and bagged trees lead to better results compared to bagged logistic regression with respect to the F1 measure.

The performance regarding the Sharpe ratio is also relatively weak. Without sample weights, bagged trees exhibit the worst results regarding maximum drawdown and Sharpe ratio.

In Figure A.4 the DAX portfolio weights are illustrated. Therefore, the backtest paths were bootstrapped to generate a 95% confidence interval. Figure A.4 shows very small DAX weights. Without considering sample weights, the logistic regression and the bagged logistic regression have more stable DAX weights. This means that the model rejects to change the portfolio. On the contrary, bagged trees and random forest have more unstable portfolio weights, and in general higher DAX weights. In Figure A.5 the cumulative performance of the respective models is compared with an investment in the DAX and the REXP respectively. In all models, the portfolio investment is highly correlated with the REXP because the DAX weights are very low. In Figure A.5 bagged trees yield the best results, when incorporating sample weights.

5.3.2 Principal Component Analysis

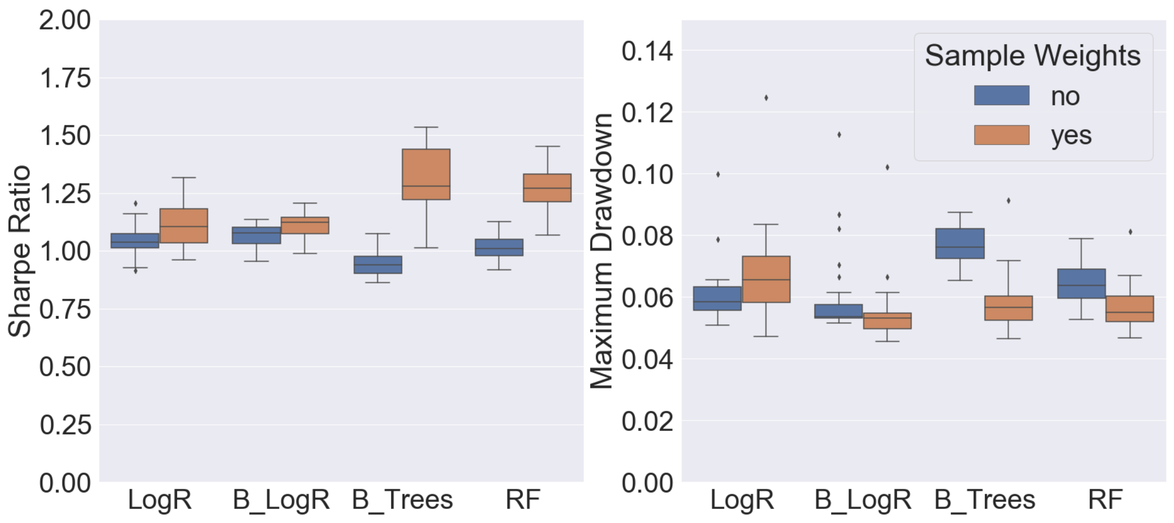


Figure 5.15: PCA: Sharpe ratio, maximum drawdown

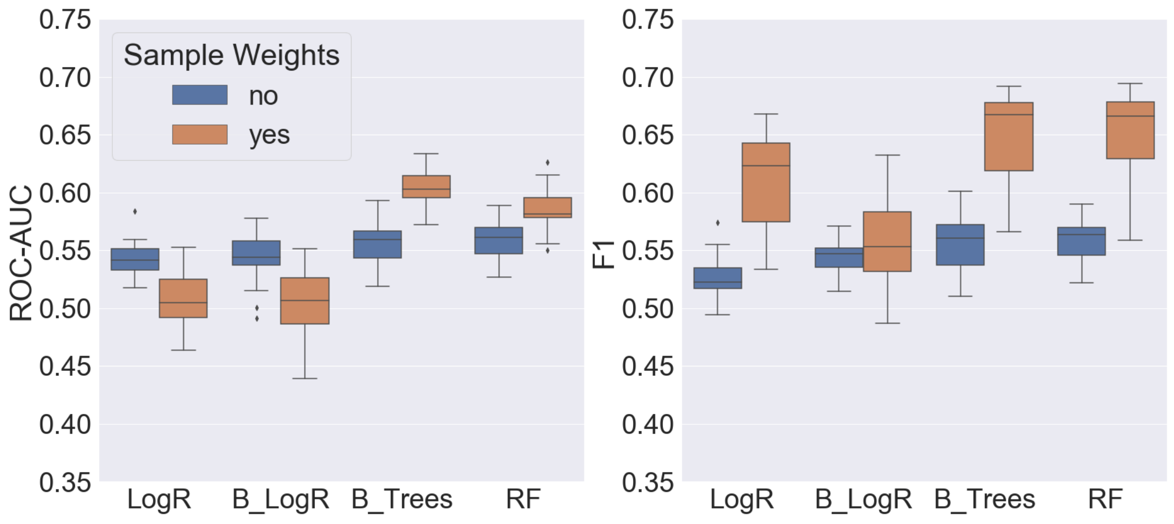


Figure 5.16: PCA: ROC-AUC, F1

In the application features are transformed by PCA. As illustrated in Figure 5.16, incorporating sample weights in bagged trees and random forest leads to better model performance results compared to logistic regression and bagged logistic regression. When considering sample weights, random forest and bagged trees have a median ROC-AUC around 0.6, which is high for stock price predicting. Regarding the F1 measure, random forest and bagged trees expose high values compared with bagged logistic regression. Without sample weights the model results are rather weak, although logistic regression and bagged logistic regression exhibit better ROC-AUC values. By including sample weights, random forest and bagged trees lead to better results regarding Sharpe ratio compared to logistic regression and bagged logistic regression. In the case of no sample weights these models score badly in comparison with logistic regression and bagged logistic regression. In Figure 5.15 the maximum drawdown results of the respective models are illustrated. Most models have outliers regarding this statistic. Without sample weights the random forest has the highest median maximum drawdown. In Figure A.6 the DAX portfolio weights are plotted. Again, the model predictions lead to very low DAX portfolio weights. The cumulative performance of the respective portfolio against DAX and REXP is illustrated in Figure A.7. Bagged trees lead to the best results in comparison with the other models.

5.3.3 Best Features

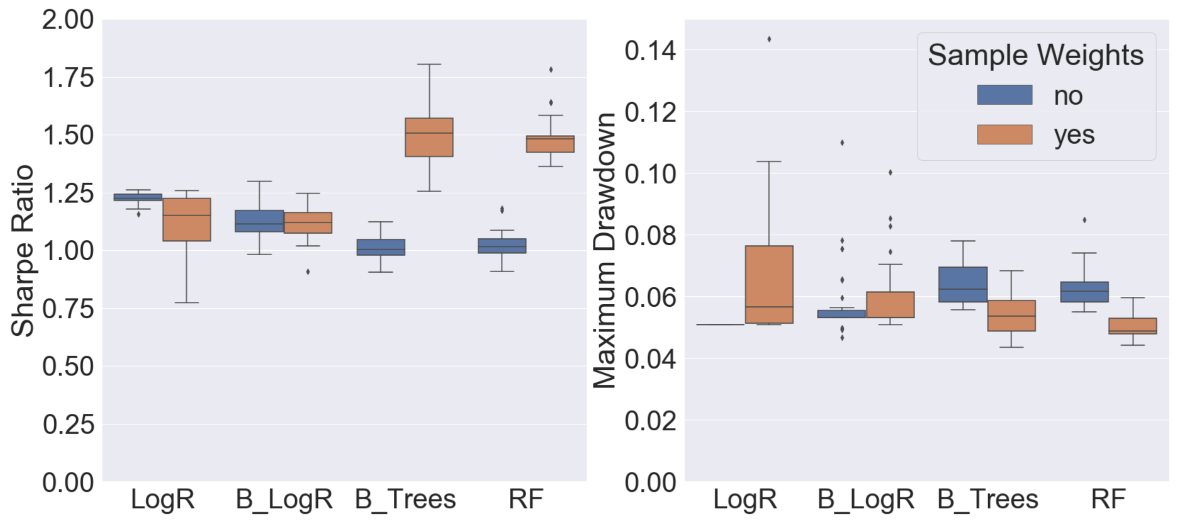


Figure 5.17: Selected features: Sharpe ratio, maximum drawdown

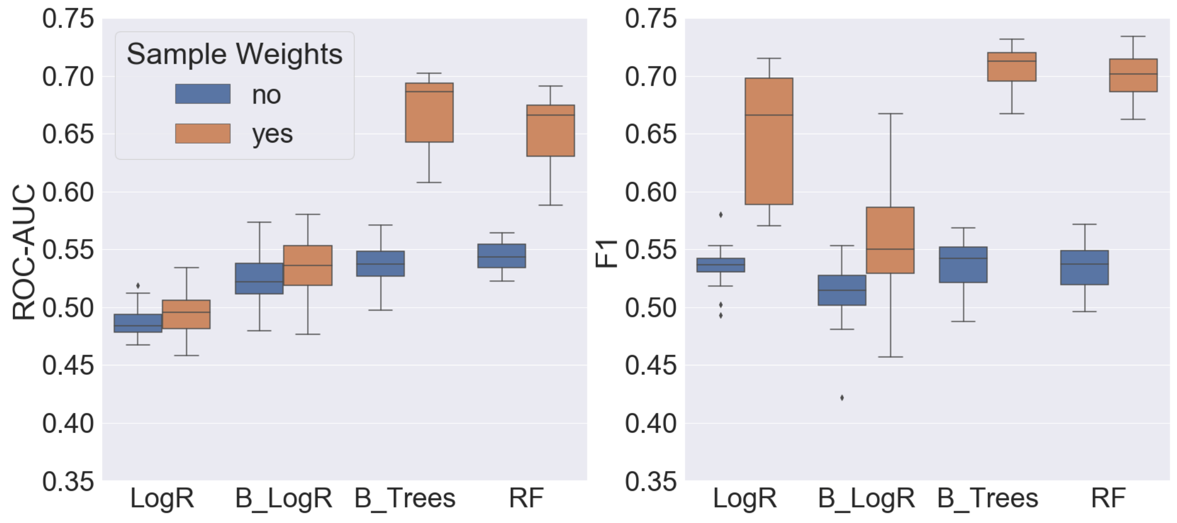


Figure 5.18: Selected features: ROC-AUC, F1

Using the insights of the research, modelling might improve by considering the best features. In Table 5.5 the selected features are listed.

In Figure 5.18 the results of the models regarding ROC-AUC and F1 measure are illustrated. Bagged trees and random forest exhibit superior model performance when incorporating sample weights. This can be observed regarding all four backtest measures.

	Without sample weights	With sample weights
features	• frac_diff_feat • momentum_20 • roll_vola_5 • log_ret_3	• log_rets • momentum_20 • roll_mean_10 • roll_mean_ewm_20

Table 5.5: Selected features

Without incorporating sample weights, the model results are rather weak. Furthermore, the logistic regression and the bagged logistic regression fail to identify patterns in the dataset. This might be caused by the collinearity of the selected features. In Figure A.8 the DAX portfolio weights are plotted. In contrast to the low portfolio weights regarding logistic regression and bagged logistic regression, random forest and bagged trees exhibit comparatively high DAX weights. The weights in these two cases vary a lot and are rather jagged. This behaviour exposes a downside of the meta-labelling approach. Since the varying portfolio weights is accompanied with high transaction costs. In Figure A.9, the cumulative performance is visible. By considering sample weights, bagged trees and random forest exhibit a very good performance. These two portfolios achieve a higher return than the only DAX portfolio. Furthermore, the risk in these two portfolios seems relatively low in comparison with the DAX index.

In general, including sample weights leads on average to far better model results, especially with random forest and bagged trees. Bagged logistic regression exhibits often worse results than applying the logistic regression alone. Applying PCA and including sample weights in random forest and bagged trees results in solid results.

6 Summary and Outlook

Predicting financial time series needs a completely different framework compared to “usual” machine learning tasks like loan predicting. An algorithm which perfectly predicts car images, could probably fail to predict stock movements. The framework in this thesis considers the specific characteristics of financial time series. The triple-barrier method incorporates the heteroskedasticity of time series by labelling observations in consideration of volatility movements. Another point to deal with is the low signal to noise ratio which is a special characteristic of financial time series. CUSUM filter offers an opportunity to deal with this issue by providing an event-based sampling methodology, though the selection of the threshold parameter is arbitrary. Another point of financial machine learning is the dependency between the observations. Most of non-financial machine learning researcher assume that the observations are drawn by an IID process. This is not valid for financial machine learning. Most labels overlap, and therefore it cannot be assured which features caused a certain outcome. Sampling weights by their absolute amount of return and their uniqueness tackles this problem.

In this thesis the upper preprocessing steps were considered to provide a dynamic investing strategy. Four models with and without sample weights were applied the the DAX dataset. Meta-labelling was applied to increase the F1-measure and learn the size of the model. To convert probabilities to bets or rather portfolio weights, a bet sizing approach as stated by López De Prado (2018) was applied. These bets in range of $[0, 1]$ were considered as portfolio weights for the DAX time series, whereas the remaining part is invested in the REXP time series. Before considering the CPCV algorithm to start backtesting, feature importance measures were calculated on the filtered dataset. Three measures with different advantages were calculated with and without sample weights. By incorporating sample weights, the features incorporating the rolling mean and the last returns were ranked as superior features to predict the patterns in the DAX time series. Without consideration of sample weights, the fractional differentiated time series was ranked high. After evaluating feature importance, the CPCV algorithm was considered for backtesting the methods. 30 paths were generated to confront the models with different scenarios. In general, the models were evaluated with different feature constellation and with the performance measures ROC-AUC, F1, Sharpe ratio and maximum drawdown. First, all features were considered for backtesting. In this application all models performed rather poorly. PCA improved the model results, especially regarding bagged trees and random forest, when sample weights were considered. Keeping only the “best” features regarding their feature importance leads to superior results. Random forest and bagged trees exhibit high Sharpe ratios and low maximum drawdowns, when incorporating sample weights. Logistic regression and bagged logistic regression were mostly not able to detect predictive patterns in the dataset.

This thesis mainly considered DAX time bars. However, as explained by López De Prado (2018), time bars have rather poor statistical properties like autocorrelation and non-normality. As stated by Mandelbrot & Taylor (1967), so called tick bars exhibits desirable statistical properties. Furthermore, volume or dollar bars could be considered for the same reasons as tick bars. Achieving returns closer to the IID assumptions could be useful because many models require these assumptions (López De Prado 2018). In further applications other models as neural networks or boosting could be tested. However, these models must be used carefully, otherwise they mistake noise for signal.

A Appendix

A.1 Proofs

- Mean-squared error decomposition

$$\begin{aligned}\mathbb{E}[(y-\hat{f})^2] &= \mathbb{E}[(y-f+f-\hat{f})] \\ &= \mathbb{E}[(y-f)^2 + (f-\hat{f})^2 + 2(y-f)(f-\hat{f})] \\ &= \sigma^2 + \mathbb{E}[(\hat{f}-f)^2] + 2\mathbb{E}[(y-f)(f-\hat{f})]\end{aligned}$$

Note that $f = f(x_0)$ and $\hat{f} = \hat{f}(x_0)$ are abbreviated. The second term can be simplified by means of following expressions:

$$\begin{aligned}\mathbb{E}[(\hat{f}-f)^2] &= \mathbb{E}[(\hat{f}-\mathbb{E}[\hat{f}]+\mathbb{E}[\hat{f}]-f)^2] \\ &= \mathbb{E}[(\hat{f}-\mathbb{E}[\hat{f}])^2] + (\mathbb{E}[\hat{f}]-f)^2 + 2\mathbb{E}[(\hat{f}-\mathbb{E}[\hat{f}])(\mathbb{E}[\hat{f}]-f)] \\ &= \mathrm{Var}[\hat{f}] + \mathrm{Bias}^2[\hat{f}] + 2\mathbb{E}[(\hat{f}-\mathbb{E}[\hat{f}])(\mathbb{E}[\hat{f}]-f)]\end{aligned}$$

These equations leads to the upper results since $\mathbb{E}[(y-f)(f-\hat{f})]$ and $\mathbb{E}[(\hat{f}-\mathbb{E}[\hat{f}])(\mathbb{E}[\hat{f}]-f)]$ equals to zero

$$\begin{aligned}\mathbb{E}[(\hat{f}-\mathbb{E}[\hat{f}])(\mathbb{E}[\hat{f}]-f)] &= \mathbb{E}\Big[\hat{f}\mathbb{E}[\hat{f}] - \mathbb{E}[\hat{f}]\mathbb{E}[\hat{f}] - \hat{f}f + \mathbb{E}[\hat{f}]f\Big] \\ &= \mathbb{E}[\hat{f}]\mathbb{E}[\hat{f}] - \mathbb{E}[\hat{f}]\mathbb{E}[\hat{f}] - \mathbb{E}[\hat{f}]f + \mathbb{E}[\hat{f}]f \\ &= 0\end{aligned}$$

$$\begin{aligned}\mathbb{E}[(y-f)(f-\hat{f})] &= \mathbb{E}[yf - f^2 - y\hat{f} + f\hat{f}] \\ &= f^2 - f^2 - \mathbb{E}[y\hat{f}] + f\mathbb{E}[\hat{f}] \\ &= -\mathbb{E}[(f+\epsilon)\hat{f}] + f\mathbb{E}[\hat{f}] \\ &= -\mathbb{E}[f\hat{f}] - \mathbb{E}[\epsilon\hat{f}] + f\mathbb{E}[\hat{f}] \\ &= 0\end{aligned}$$

The last equation $\mathbb{E}[\epsilon\hat{f}]$ equals to zero, because ϵ and $\hat{f}$ are independent since ϵ on the new data point x_0 is independent of the other data points which were used to create $\hat{f}$ (Hyndman 2015).

- Maximum value

In this proof a derivation for a sample maximum $\max_s = \max\{z_s\}$ should be derived, as stated by Bailey *et al.* (2014). Supposing a sample of $s = 1, \dots, \mathcal{S}$ IID random variables and $z_s \sim \Phi$, whereas Φ is the CDF of the standard normal distribution. Applying the Fisher-Tippett-Gnedenko theorem to the gaussian distribution leads to

$$\lim_{\mathcal{S}\to\infty}\mathbb{P}[\frac{\max_{\mathcal{S}} - \alpha}{\beta} \leq x] = G(x),$$

whereas $G(x) = \exp(-\exp(-x))$ represents the CDF of the Standard Gumbel distribution. Furthermore, the normalizing constants β and α are defined by: $\alpha = \Phi^{-1}[1 - \frac{1}{\mathcal{S}}]$, $\beta = \Phi^{-1}[1 - \frac{1}{\mathcal{S}}\exp(-1)] - \alpha$. Φ^{-1} corresponds to the inverse of the standard Normal's CDF (Embrechts *et al.* 1997). In addition, the limits of a normalized maxima are defined by

$$\lim_{\mathcal{S}\to\infty}\mathbb{E}[\frac{\max_{\mathcal{S}} - \alpha}{\beta}] = \gamma,$$

where γ represents the Euler-Mascheroni constant. Thus, for a sufficient large $\mathcal{S}$ the expected maximum of a standard normal distribution can be expressed by

$\approx \alpha + \beta\gamma = (1-\gamma)\Phi^{-1}[1 - \frac{1}{\mathcal{S}}] + \gamma\Phi^{-1}[1 - \frac{1}{\mathcal{S}}\exp(-1)]$ (Bailey *et al.* 2014).

A.2 Figures

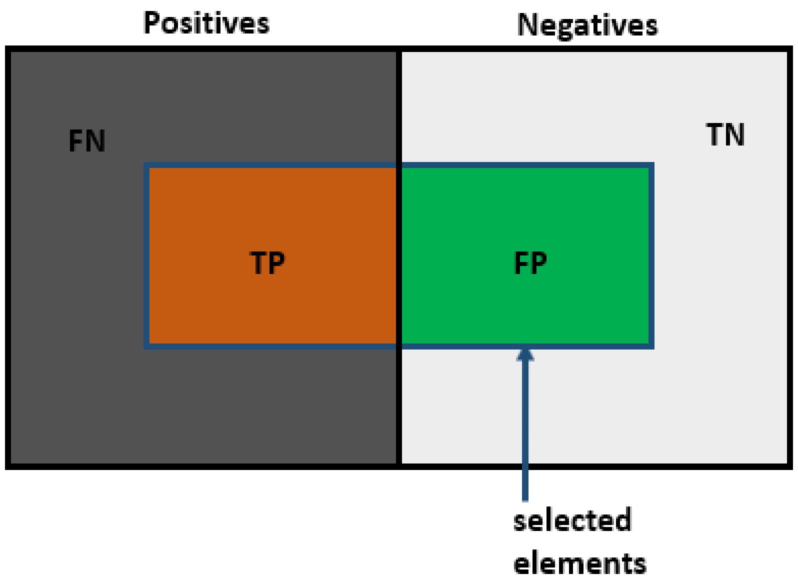


Figure A.1: Confusion matrix

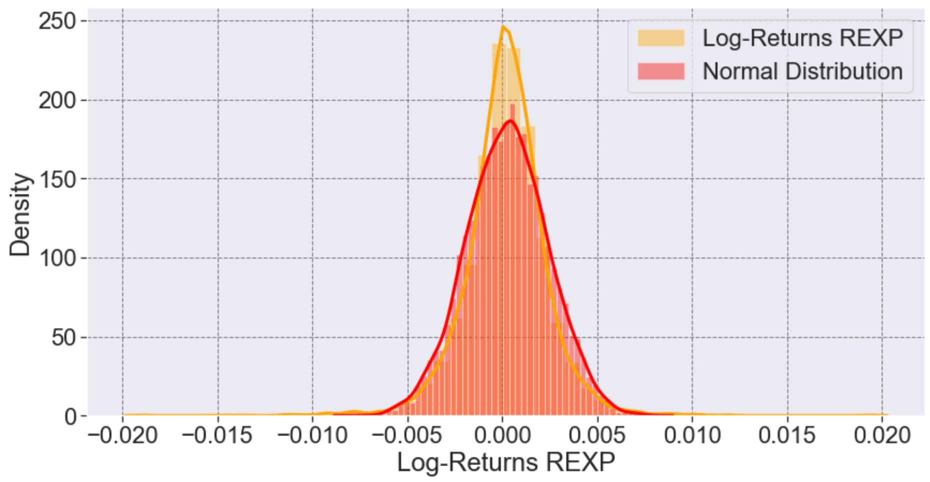


Figure A.2: Histogram REXP

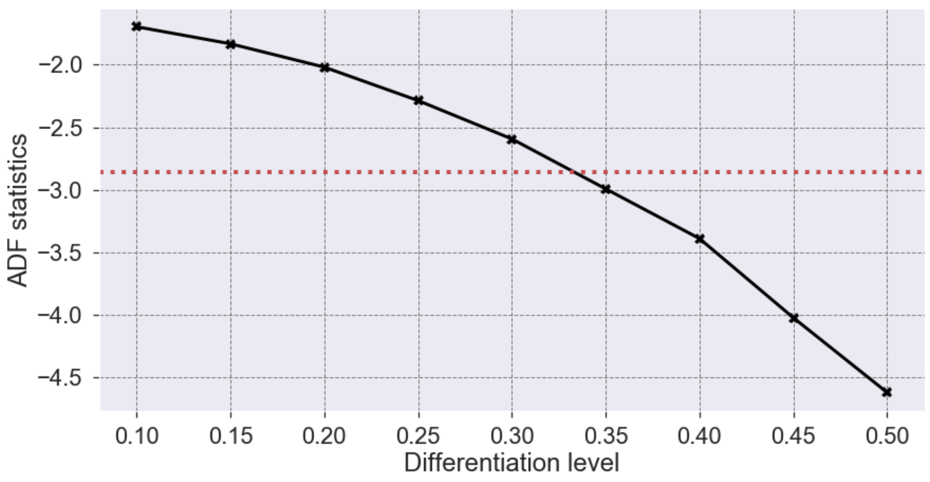


Figure A.3: Test statistic for different d values

- All features

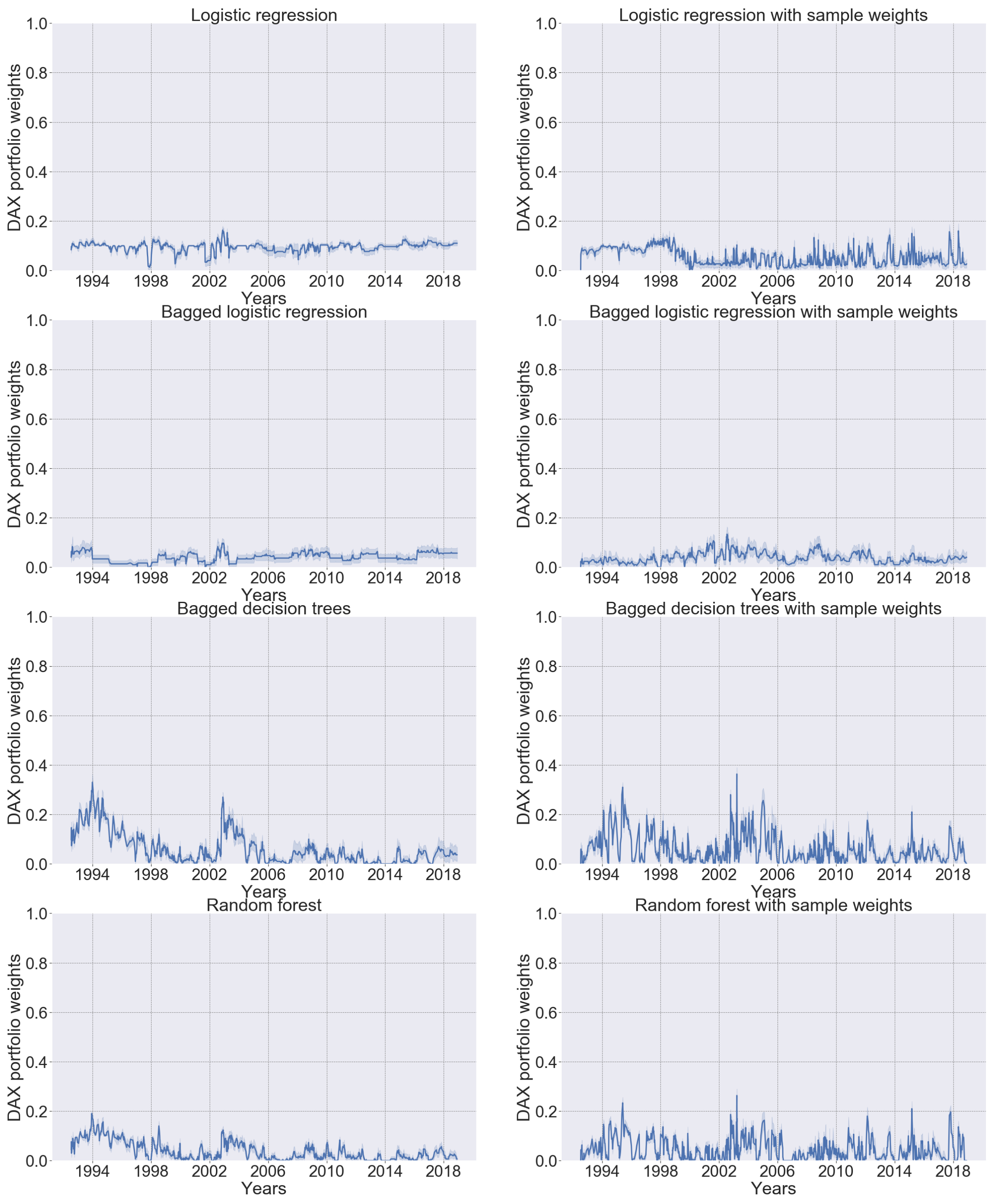


Figure A.4: DAX portfolio weights, all features

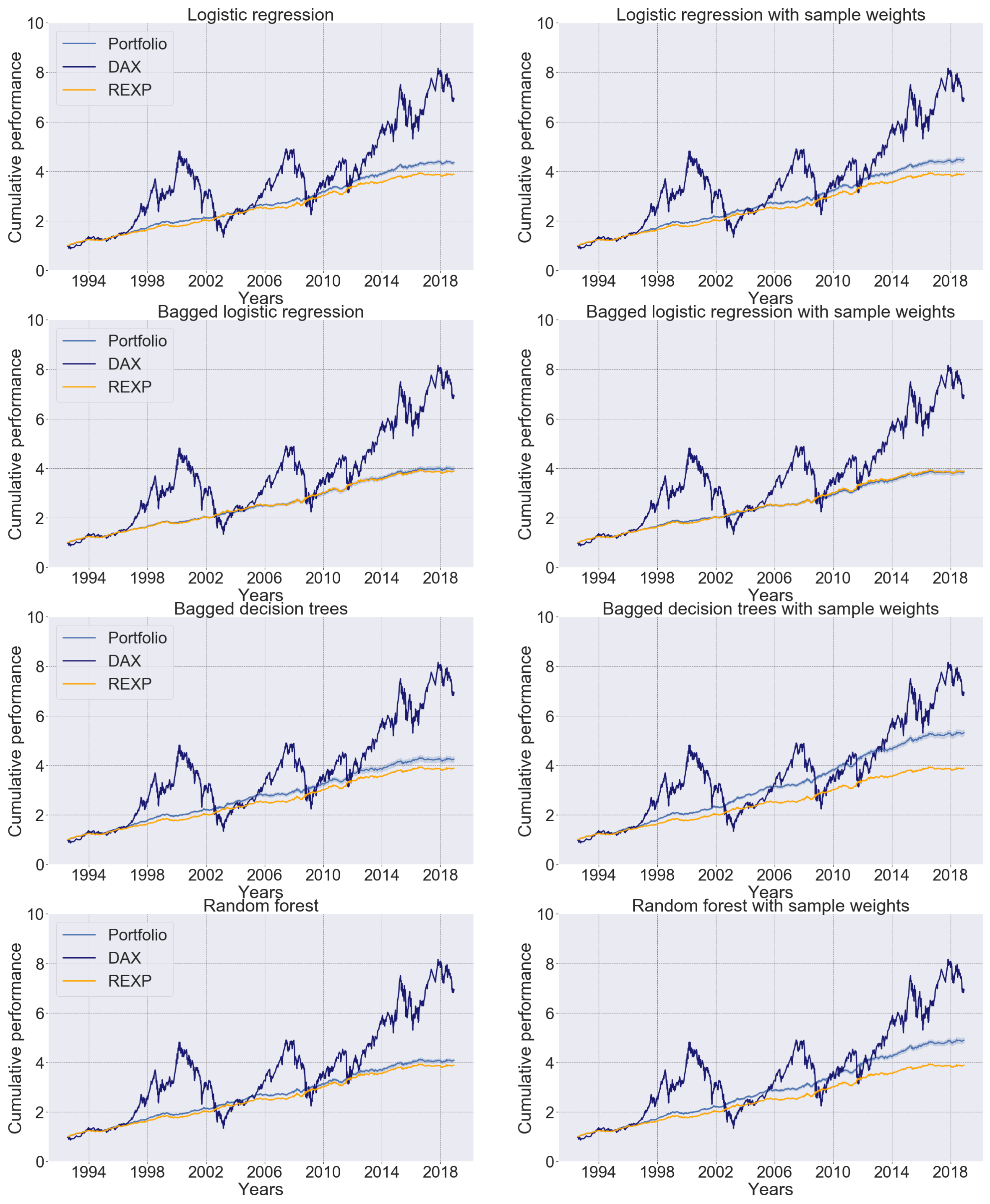


Figure A.5: Cumulative performance, all features

- PCA

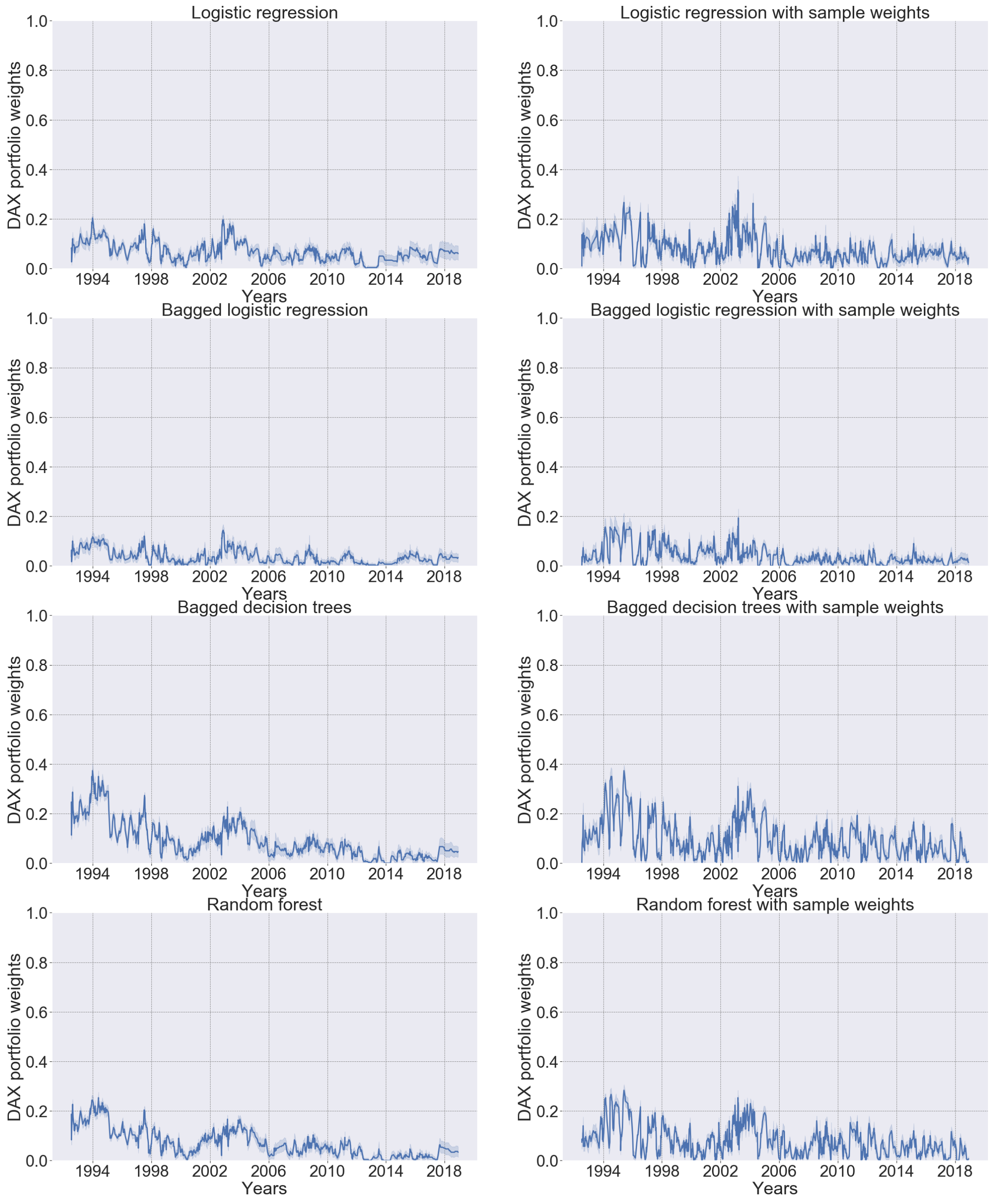


Figure A.6: DAX portfolio weights, PCA

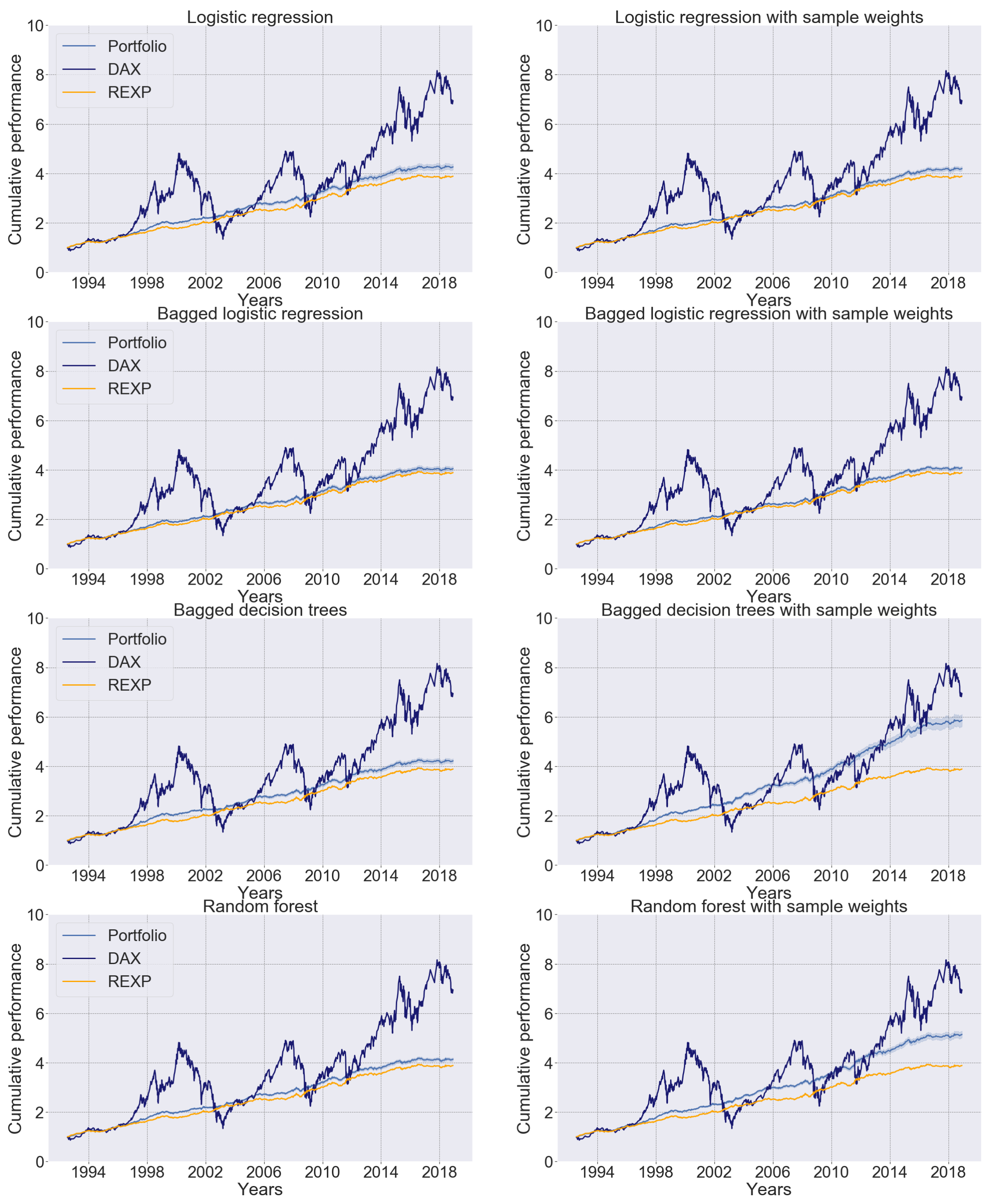


Figure A.7: Cumulative performance, PCA

- Best features

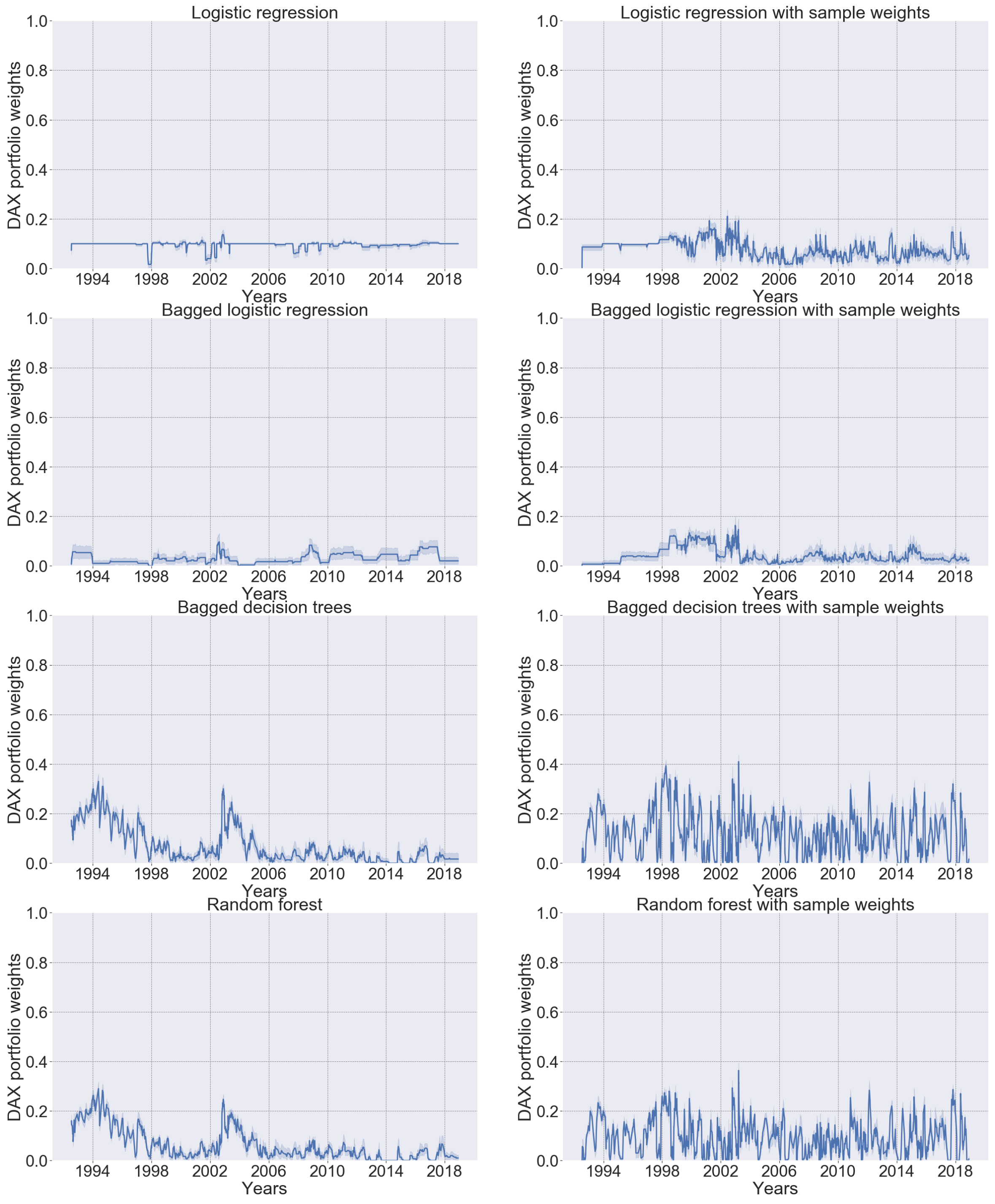


Figure A.8: DAX portfolio weights, best features

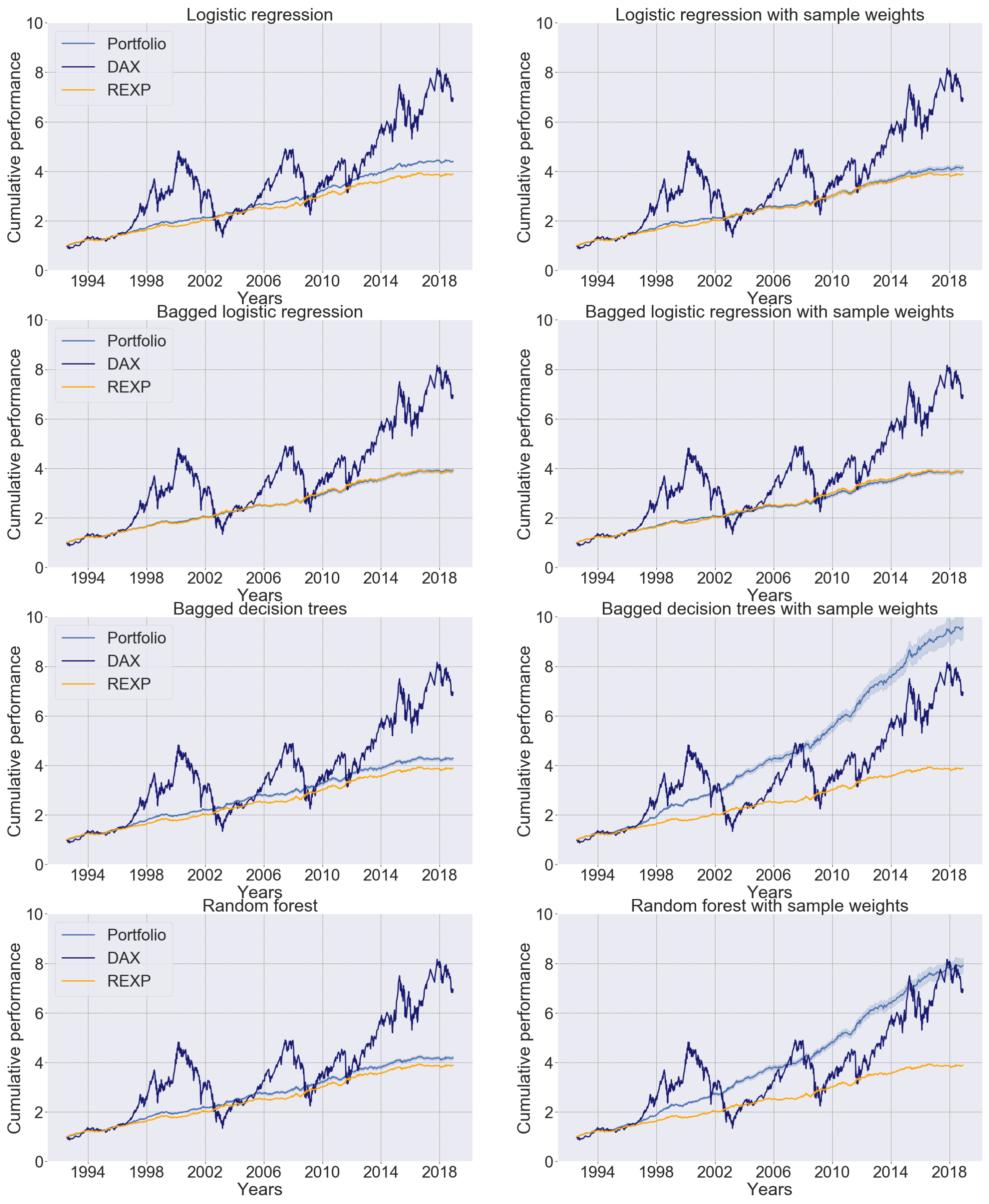


Figure A.9: Cumulative performance, best features

B Electronic Appendix

The electronic appendix consists of the *classes* folder, *Simulations* folder, *snippets.py* script and the *document* folder.

- Classes folder
  - Preprocessing_class: Contains the code of the preprocessing class. Thus, the computation of the sample weights and the fracdiff series are included
  - CPCV_backtest_algorithm: Creates a CPCV object which is used for the backtesting

- Simulations folder
  - Comprises three folders, all_feats, pca and best_feats. All of the three consists of a python script and the results as PKL data respectively
  - graphic_script: Produces the graphics out of the PKL data
  - descriptive_analysis: Includes the code to create all graphics in the thesis with the exception of the figures in subsection 5.3

- Snippets: Contains several support functions to enable the two main classes working

- Document folder
  - Masterthesis: The thesis as PDF document
  - Image folder: Comprises all images illustrated in the thesis
  - Data folder: Contains DAX data (retrieved April 6, 2019, from https://stooq.com/q/d/?s=^dax) and REXP data (retrieved May 22, 2019, from https://www.ariva.de/rex_performance-index/historische_kurse?boerse_id=12&month=&clean_split=1&clean_split=0&clean_payout=0&clean_bezug=1&clean_bezug=0; missing data at August 26, 2013 & March 12, 2013 replaced with data from https://www.finanzen.net/zinsen/historisch/rex-performance-index, retrieved May 22, 2019)

The simulations and figures were computed with Python 3.7.0 (Van Rossum & Drake 2011) and Spyder 3.3.4. (Raybaut & Cordoba 2009). The considered packages are listed below with the respective version number and source.

- Cython 0.29.4 (Behnel *et al.* 2011)
- matplotlib 2.2.3 (Hunter 2007)
- multiprocess 0.70.7 (McKerns *et al.* 2012)
- numba 0.39.0 (Lam *et al.* 2015)
- numpy 1.15.4 (Walt *et al.* 2011)
- Pandas 0.23.4 (McKinney 2010)
- scikit-learn 0.19.2 (Pedregosa *et al.* 2011)
- scipy 1.1.0 (Jones *et al.* 2001)
- seaborn 0.9.0 (Waskom *et al.* 2014)
- statsmodels 0.9.0 (Seabold & Perktold 2010)
- tqdm 4.26 (Costa-Luis 2019)

C References

Alexander, C. (2001). *Market models: A guide to financial data analysis.* John Wiley & Sons, Chichester.

Asmundsson, J. (2018). The Big Problem With Machine Learning Algorithms. Retrieved June 2, 2019, from https://www.bloomberg.com/news/articles/2018-10-09/the-big-problem-with-machine-learning-algorithms.

Bailey, D.H. & Lopez de Prado, M. (2014). *The Deflated Sharpe Ratio: Correcting for Selection Bias, Backtest Overfitting and Non-Normality.* Social Science Research Network, Rochester, NY.

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**, 458.

Bao, W., Yue, J. & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory (B. Podobnik, Ed.). *PLOS ONE*, **12**, e0180944.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D.S. & Smith, K. (2011). Cython: The Best of Both Worlds. *Computing in Science & Engineering*, **13**, 31–39.

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

Breiman, L. (2001). Random Forests. *Mach. Learn.*, **45**, 5–32.

Breiman, L., Friedman, J., Stone, C.J. & Olshen, R.A. (1984). *Classification and Regression Trees.* Taylor & Francis.

Costa-Luis, C.O. da. (2019). Tqdm: A Fast, Extensible Progress Meter for Python and CLI.

Embrechts, P., Klüppelberg, C. & Mikosch, T. (1997). *Modelling Extremal Events: For Insurance and Finance.* Springer-Verlag, Berlin Heidelberg.

Fahrmeir, L., Kneib, T. & Lang, S. (2009). *Regression: Modelle, Methoden und Anwendungen*, 2nd edn. Springer-Verlag, Berlin Heidelberg.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, **27**, 861–874.

Hamelink, F. & Hoesli, M. (2004). Maximum drawdown and the allocation to real estate. *Journal of Property Research*, **21**, 5–29.

Hastie, T., Tibshirani, R. & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, 2nd edn. Springer-Verlag, New York.

Hosking, J.R.M. (1981). Fractional Differencing. *Biometrika*, **68**, 165–176.

Hunter, J.D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering*, **9**, 90–95.

Hyndman, R.J. (2015). Bias variance decomposition. 1.

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013). *An Introduction to Statistical Learning: With Applications in R*. Springer-Verlag, New York.

Jones, E., Oliphant, T. & Peterson, P. (2001). SciPy: Open Source Scientific Tools for Python.

Lam, K. & Yam, H.C. (1997). Cusum Techniques for Technical Trading in Financial Markets. *Financial Engineering and the Japanese Markets*, **4**, 257–274.

Lam, S.K., Pitrou, A. & Seibert, S. (2015). Numba: A LLVM-based Python JIT Compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 7:1–7:6. LLVM '15. ACM, New York, NY, USA.

Lo, A.W. (2002). The Statistics of Sharpe Ratios. *Financial Analysts Journal*, **58**, 36–52.

Louppe, G., Wehenkel, L., Sutera, A. & Geurts, P. (2013). Understanding variable importances in forests of randomized trees. 9.

López De Prado, M. (2018). *Advances in Financial Machine Learning*. John Wiley & Sons, New Jersey.

MacKinnon, J.G. (2010). *Critical Values For Cointegration Tests*. Economics Department, Queen's University.

Mandelbrot, B. & Taylor, H.M. (1967). On the Distribution of Stock Price Differences. *Operations Research*, **15**, 1057–1062.

Manning, C., Raghavan, P. & Schuetze, H. (2009). Introduction to Information Retrieval. 581.

McKerns, M.M., Strand, L., Sullivan, T., Fang, A. & Aivazis, M.A.G. (2012). Building a Framework for Predictive Science. *arXiv:1202.1056 [cs].*

McKinney, W. (2010). Data Structures for Statistical Computing in Python. pp. 51–56.

Page, E.S. (1954). Continuous Inspection Schemes. *Biometrika*, **41**, 100–115.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, **12**, 2825–2830.

Raschka, S. & Mirjalili, V. (2017). *Machine Learning mit Python und Scikit-Learn und TensorFlow: Das Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning*. mitp, Frechen, Germany.

Raybaut, P. & Cordoba, C. (2009). Spyder Website. Retrieved, June 03, 2019, from https://www.spyder-ide.org/.

Ruppert, D. & Matteson, D.S. (2015). *Statistics and Data Analysis for Financial Engineering: With R examples*, 2nd edn. Springer-Verlag, New York.

Seabold, S. & Perktold, J. (2010). Statsmodels: Econometric and Statistical Modeling with Python. pp. 57–61.

Sharpe, W.F. (1966). Mutual Fund Performance. *The Journal of Business*, **39**, 119–138.

Sharpe, W.F. (1994). The Sharpe Ratio. *The Journal of Portfolio Management*, **21**, 49–58.

Van Rossum, G. & Drake, F.L. (2011). Python Software Foundation. Retrieved June 2, 2019, from https://www.python.org/psf-landing/. *Python.org*.

Walt, S. van der, Colbert, S.C. & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, **13**, 22–30.

Waskom, M., Botvinnik, O., Hobson, P., Cole, J.B., Halchenko, Y., Hoyer, S., Miles, A., Augspurger, T., Yarkoni, T., Megies, T., Coelho, L.P., Wehner, D., cynddl, Ziegler, E., diego0020, Zaytsev, Y.V., Hoppe, T., Seabold, S., Cloud, P., Koskinen, M., Meyer, K., Qalieh, A. & Allan, D. (2014). Seaborn: V0.5.0 (November 2014).

Zangari, P. (1996). RiskMetrics Technical Document - Fourth Edition 1996, December.

296.

Zhou, Z.-H. (2012). *Ensemble Methods: Foundations and Algorithms*, 1st edn. Chapman; Hall/CRC.

Declaration

Signature	Date