LEVERAGING TECHNICAL AND FUNDAMENTAL DATA WITH MACHINE LEARNING ALGORITHMS FOR EMISSIONS TRADING

Álvaro García Adrados

Trabajo de investigación 25/003

Master en Banca y Finanzas Cuantitativas

Tutores: Dr. Álvaro Montealegre
D. Javier Pérez
Dr. Manuel Moreno

Universidad Complutense de Madrid

Universidad del País Vasco

Universidad de Valencia

Universidad de Castilla-La Mancha

www.finanzascuantitativas.com

LEVERAGING TECHNICAL AND FUNDAMENTAL DATA WITH MACHINE LEARNING ALGORITHMS FOR EMISSIONS TRADING

ÁLVARO GARCÍA ADRADOS

FACULTAD DE CIENCIAS ECONÓMICAS Y EMPRESARIALES
UNIVERSIDAD COMPLUTENSE DE MADRID

Máster en Banca y Finanzas Cuantitativas

18-06-2025

Director/es y/o colaborador:

Dr. Álvaro Montealegre (Repsol)
Javier Pérez (Repsol)
Dr. Manuel Moreno (UCLM)

Resumen

Este estudio analiza cómo la utilización de algoritmos de aprendizaje automático puede mejorar las estrategias de negociación en el mercado de futuros sobre derechos de emisión de la Unión Europea (EUA), combinando el análisis técnico con datos fundamentales del mercado energético. A través del uso de indicadores técnicos bien conocidos como la media móvil simple (SMA), el índice de fuerza relativa (RSI), el MACD y las bandas de Bollinger, junto con variables como los precios de futuros sobre el gas, carbón y electricidad, este trabajo intenta mejorar la comprensión del comportamiento complejo de este mercado volátil. Se han aplicado y validado cuidadosamente técnicas como *Random Forest* y *XGBoost*, aplicando técnicas de preprocesamiento y validación cruzada para evitar el sobreajuste y asegurar resultados fiables.

Los resultados muestran que *Random Forest* y *XGBoost* logran una precisión muy alta en los datos de entrenamiento y ofrecen señales sólidas y consistentes al aplicarse sobre datos nuevos. Al incorporar estas técnicas en estrategias de negociación automatizada, estas señales superan claramente a enfoques tradicionales como "comprar y mantener" o aquellos basados exclusivamente en indicadores técnicos. La inclusión de variables fundamentales—como los diferenciales de generación eléctrica (*Dark* y *Clean Spark Spread*) y el índice bursátil EURO STOXX 50—aporta un contexto macroeconómico valioso. Además, las diferencias en la importancia de las variables entre modelos sugieren que cada uno capta patrones distintos, lo que abre la posibilidad de diseñar estrategias híbridas basadas en su consenso. En resumen, esta investigación ha mostrado el potencial real del aprendizaje automático para respaldar decisiones de inversión más inteligentes y adaptativas en los mercados de emisiones de carbono.

Palabras clave

Negociación de emisiones, Aprendizaje automático, Análisis técnico, Análisis fundamental, Futuros sobre EUA, *Random Forest*, *XGBoost*.

Abstract

This study analyzes how machine learning (ML) can be used to improve trading strategies in the European Union Allowances (EUA) futures market by combining technical analysis with fundamental energy market data. By using well-known technical indicators like the Simple Moving Average (SMA), Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), and Bollinger Bands, alongside variables such as gas, coal, and electricity futures prices, this study aims to improve our understanding of the complex behavior of this volatile market. ML techniques such as Random Forest and XGBoost were applied and carefully validated using preprocessing and cross-validation techniques to avoid overfitting and ensure reliable results.

The results show that, although both Random Forest and XGBoost perform very adequately on training data, they also provide solid and consistent signals when tested on new data. When applied to automated trading strategies, these signals outperform more traditional approaches like Buy and Hold or those based only on technical indicators. Including fundamental variables—such as the Dark and Clean Spark Spreads and the EURO STOXX 50 index—adds valuable macroeconomic context. Interestingly, the models differ in how they prioritize features, suggesting that each one captures different aspects of the market, which opens the door to building hybrid strategies based on their agreement. In short, this research has shown the real potential of ML to support smarter and more adaptive trading decisions in carbon markets.

Keywords

Emissions Trading, Machine Learning, Technical Analysis, Fundamental Analysis, EUA Futures, Random Forest, XGBoost.

Contents

1. Introduction

1.1 Background and General Context

Emissions trading has become an essential mechanism for combating climate change, providing incentives for reducing greenhouse gases through market instruments. The complexity of financial and energy markets requires the adoption of analytical tools to anticipate the price movements of derivatives on emission allowances (in this case, futures). This study addresses this challenge by integrating technical and fundamental analysis data into machine learning (ML) models, with the aim of designing automated trading strategies that optimize decision-making.

This thesis is based on two fundamental pillars. On the one hand, key technical indicators are analyzed — including the Simple Moving Average (SMA), the Relative Strength Index (RSI), the Moving Average Convergence Divergence (MACD), and Bollinger Bands — which provide a clear understanding of market trends and volatility. On the other hand, fundamental variables from energy markets, such as gas, coal, and electricity futures, are incorporated, complemented by macroeconomic indicators such as the EURO STOXX 50 index. This combination allows us to address both the technical aspects and the underlying economic aspects that influence the behavior of emission prices.

Our methodology includes data preprocessing steps, such as the elimination of erroneous records, the use of rolling techniques (to obtain a single series of prices) and the unification of the different sources of information. Cross-validation is also used to ensure that the models learn generalizable patterns and do not compromise the predictive capacity of new data. Random Forest and XGBoost algorithms are used to classify the performance of EUA futures returns into distinct categories, allowing the design of a strategy that assigns long or short positions according to the signal generated.

In short, this study aims to demonstrate that the fusion of ML techniques with technical and fundamental analysis not only improves the accuracy in predicting returns, but also offers more adaptive and robust trading strategies than other more classical ones in carbon markets. In this way, it aims to open new lines of research and practical applications in the field of ML for emissions trading.

1.2 Overview of the Emissions Market

Emissions trading systems are market-based instruments designed to reduce greenhouse gas (GHG) emissions. In these systems, a regulator sets a maximum limit on the allowed emis-

sions for all participants. At the end of each defined period, participants (such as facilities, airline operators, and individuals) must surrender a number of emission allowances equal to their emissions during that period. These allowances are permits that allow their holder to release one ton of carbon dioxide (CO_2) and can be traded (bought or sold) within the market [1].

The idea of emissions trading first emerged in the 1960s when economists suggested using market mechanisms to control pollution. This system became a key reference for the development of the European Union Emissions Trading System (EU ETS), which was officially launched in 2005 [2].

The EU ETS is the world's largest carbon market, covering about 40% of total greenhouse gas emissions in the European Union [3]. It operates under a cap-and-trade system, which works as follows:

1. An upper limit (cap) of emissions that can be emitted is defined.
2. Companies receive or buy emission allowances (EUA - European Union Allowances) that allow them to emit a certain amount of CO_2.
3. Companies that reduce their emissions below their allowances can sell their allowances to others who need more.

The main objective is to reduce the EU's greenhouse gas emissions by 55% from 1990 levels by 2030.

The EU ETS operates in compliance phases:

- **Phase 1 (2005-2007)**: Learning period with free allocation of allowances. It served to set the price of emission allowances.
- **Phase 2 (2008-2012)**: Linked to the Kyoto Protocol,[1] introducing stricter emission reductions. Free allocation was reduced, and some countries began auctioning allowances.
- **Phase 3 (2013-2020)**: Auctions became the main method for distributing allowances, and many free allocations were eliminated. A single EU-wide emissions cap was introduced, unlike in the previous phases.
- **Phase 4 (2021-2030)**: Stronger alignment with the European Green Deal's climate goals, aiming for a 62% emissions reduction by 2030 compared to 2005.

[1] This is a protocol of the United Nations Framework Convention on Climate Change, and an international agreement aimed at reducing emissions of six greenhouse gases.

The system has helped reduce emissions, but it has faced challenges. In the first phase, too many allowances were issued, causing EUA prices to drop [4].

Over time, the reduction of the number of permits has made the system more effective. However, there are still worries about carbon leakage, where companies move to countries with weaker environmental rules to avoid stricter limits. To address this, the EU is implementing the Carbon Border Adjustment Mechanism (CBAM), which will impose a tax on imports of high-emission products from countries with weaker environmental standards [3].

The future of the emissions market is moving toward a low-carbon economy and expanding to sectors like shipping and aviation [3].

The thesis is organized as follows. Section 2 presents the literature review, where we discuss previous research on machine learning applications in emissions trading and the integration of technical and fundamental analyses. Section 3 details the data used in the study, including the EUA futures series and the fundamental variables derived from energy markets and the EURO STOXX 50 index. In Section 4, we describe the methodology: first, we explain the machine learning techniques—specifically, Random Forest and XGBoost—and then we introduce the technical indicators employed, such as SMA, RSI, MACD, and Bollinger Bands. Section 5 outlines the results, starting with the evaluation of the predictive performance of the models and followed by the performance of the trading strategies based on these predictions. Finally, Section 6 provides the conclusions, discussing the findings, implications, and potential avenues for future research.

2. Literature Review

The use of ML in financial markets has grown significantly in recent years. For example, in [5] they apply a Random Forest model to predict errors in analysts' forecasts. However, in [6], they identify that anticipation bias significantly affects the results and conclude that linear models can be equally effective in predicting and generating profits in trading. Therefore, we must be careful when working with ML models.

In [7], it is noted that the integration of alternative data with advanced modeling techniques is transforming quantitative trading. The combination of statistical knowledge, computational skills, and investment experience is key to effective predictive models and profitable trading strategies.

The European Union Emissions Trading System (EU ETS) has been the subject of research in the context of deep learning. A recent study uses LSTM neural networks [2] to predict the price of EUA futures [8]. The long-short trading strategy based on these predictions outperforms a completely random model, showing high risk-adjusted returns. In addition, the approach exhibits low correlation with traditional indices, suggesting its potential for diversification of investment portfolios.

ML algorithms, in this case Random Forest, used in trading strategies have been evaluated in multiple studies. An analysis in the Italian market investigated the ability of these models to predict returns in energy stocks [9]. Technical indicators and macroeconomic variables were used to feed the model, outperforming traditional buy-and-hold strategies. The improved version of the Random Forest-based strategy doubled returns compared to conventional approaches.

In methodological terms, the design of ML-based trading agents requires a series of well-defined steps. [10] presents a systematic framework for building predictive models, including data preprocessing, time series segmentation, dimensionality reduction, and clustering. Their proposal emphasizes the use of Random Forest as a key tool for price prediction and decision-making in automated trading.

In sum, previous studies have shown that ML-based models can generate effective trading signals and improve risk-adjusted returns. In other words, combining fundamental and technical variables with advanced models can provide significant advantages in investment decision-making.

[2] **LSTM (Long Short-Term Memory) neural networks** are a type of recurrent neural network (RNN) designed to model sequential data by capturing long-term dependencies.

3. Data

This section describes the data used in this thesis. We consider two types of data:

- Prices of EUA futures contracts (MOZ [3]), being the sample period from July 2017 to February 2025. We will analyze the futures contracts that expire in December of each year.
- Fundamental variables related to EUA futures contracts. These variables include the prices of futures contracts with expiration in the following month for several underlying assets: TTF gas (TZ1), API2 coal (HDD1), and electricity in Germany (DET1). We also added the EURO STOXX 50 index to analyze any market correlations. All the data are obtained from the Bloomberg platform and have been imported and processed using the pandas library in Python.

3.1 EUA Future Prices

The original dataset consists of futures contracts expiring in December of each year, from July 2017 to December 2025. We used a rolling strategy to get a continuous pricing series.

A rolling method is a common approach in time series analysis. In our case, we apply the rolling method by sticking with the first available contract and only switching to the next one once the current contract expires. This way, we maintain an uninterrupted price series while guaranteeing that every price point represents the most pertinent contract at that precise moment.

Figure 3.1 illustrates the resulting EUA future price series. At first, prices were low and stable. But starting in 2020, they began to rise significantly, especially around 2021-2022. There were some sharp changes, with prices even going above 100€ before dropping to a lower level. Recently, prices have started rising again. This could be due to higher costs related to CO_2 emissions, stricter regulations, or a higher demand for carbon credits.

To better understand how prices are changing, we calculate the 5-day logarithmic returns. We do this because our model predicts price movements for the upcoming week. For a given period, the logarithmic return is calculated as follows:

$$r_t = \ln\left(\frac{P_t}{P_{t-5}}\right) \tag{3.1}$$

[3]MOZ is the Bloomberg ticker symbol used to identify EUA futures contracts that expire in December of each year.

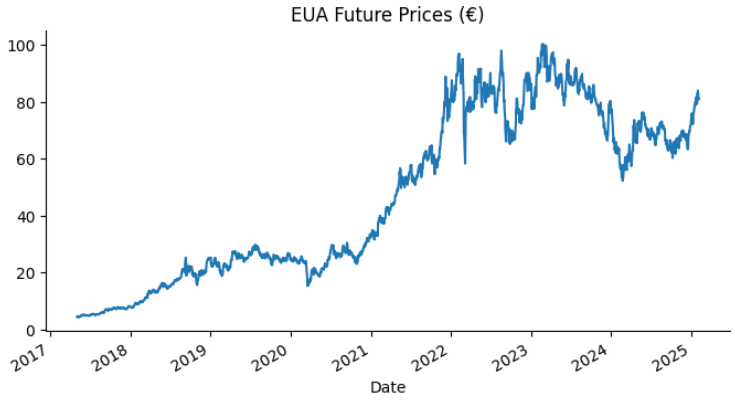

Figure 3.1: *EUA Futures Prices from 2017 to 2025.*

where P_t is the price at time t, and P_{t-5} is the price 5 days earlier. This method helps to normalize the data, making it easier to capture relative price changes over time.

Figure 3.2 displays the computed 5-day logarithmic returns. The series shows frequent fluctuations around zero, with occasional large spikes indicating periods of high volatility. The presence of extreme values suggests the influence of market shocks or external events on CO_2 futures prices.

Figure 3.2: *5-Day Logarithmic Returns of EUA Futures Prices.*

3.2 Fundamental Variables

Fundamental analysis is typically used to value a certain asset or company, and it allows one to determine whether the asset under analysis is over- or under-valued by the market. This analysis can consider the economic, market, sector-specific, and financial performance. In the context of EUA futures contracts, fundamental variables represent external factors that

affect the supply and demand for emission allowances futures. These factors often include macroeconomic data and energy pricing.

We consider the following fundamental variables:

- **TTF Gas Front Month (TZ1)**: This is the price of the TTF gas front-month futures contract, which is an important benchmark for European natural gas pricing. Natural gas is frequently used for electricity generation, especially during cold periods, when heating demand increases. The cost of emissions is influenced by changes in gas prices since gas burning releases CO_2.
- **API2 Coal Front Month (HDD1)**: This is the price of the front-month futures for API2, reference coal price in Europe. Coal power plants emit a lot of CO_2, so when they are more profitable, more emission allowances are needed.
- **German Electricity Front Month (DET1)**: This is the price of front-month electricity futures in Germany. As one of Europe's largest electricity markets, price changes reflect CO_2 emissions. Power plants use fossil fuels, so they emit CO_2 and need to buy emission allowances to cover these emissions.
- **EURO STOXX 50 Index (SX5E** [4]**)**: This index shows how the 50 major companies in the eurozone are performing, and hence gives an idea of the overall economy. When the economy grows, industries use more energy, which can increase CO_2 emissions.

Several forms of the EURO STOXX 50 Index were tested before choosing the final input for our model. We tried weekly changes, the raw index level, a 30-day moving average, and also tested the model without including the index at all. In the end, the raw index level produced consistently better results. This decision is based on the idea that emissions markets tend to react to the current absolute level of economic activity—as directly reflected in the index—rather than to smoothed, lagging signals like moving averages.

To better understand the relationship between energy markets and the cost of emissions, we calculated two key indicators of profitability in electricity generation: the Dark Spark Spread and the Clean Spark Spread. These show how profitable it is to produce electricity with natural gas and coal, respectively, taking into account the cost of CO_2 emissions. In more detail, we have the following:

- **Dark Spark Spread (DSS):** Indicates the profit margin in coal-fired electricity generation, considering the cost of coal and the price of CO_2 emissions.
- **Clean Spark Spread (CSS):** Measures the profitability of electricity generation with natural gas, including the price of gas and the cost of CO_2 emissions.

[4]TZ1, HDD1, DET1, and SX5E are the Bloomberg tickers associated with the respective instruments. From this point onward, these tickers will be used as shorthand references to denote the corresponding products.

They are calculated as follows:

$$DSS = P_{DET1} - \frac{1}{6.978} P_{HDD1} - 0.757\, P_{CO2} \tag{3.2}$$

$$CSS = P_{DET1} - 2\, P_{TZ1} - 0.36\, P_{CO2} \tag{3.3}$$

where

- P_{DET1}: German electricity futures price
- P_{HDD1}: API2 coal futures price
- P_{TZ1}: TTF gas futures price
- P_{CO2}: EUA futures price

The conversion factors in these indicators adjust coal, gas, and emissions prices so that everything is expressed in €/MWh of electricity produced. The values of the conversion factors for both formulas have been obtained from [11].

Figure 3.3 provides the correlation matrix between the weekly logarithmic returns for the four variables and the SXSE Index.

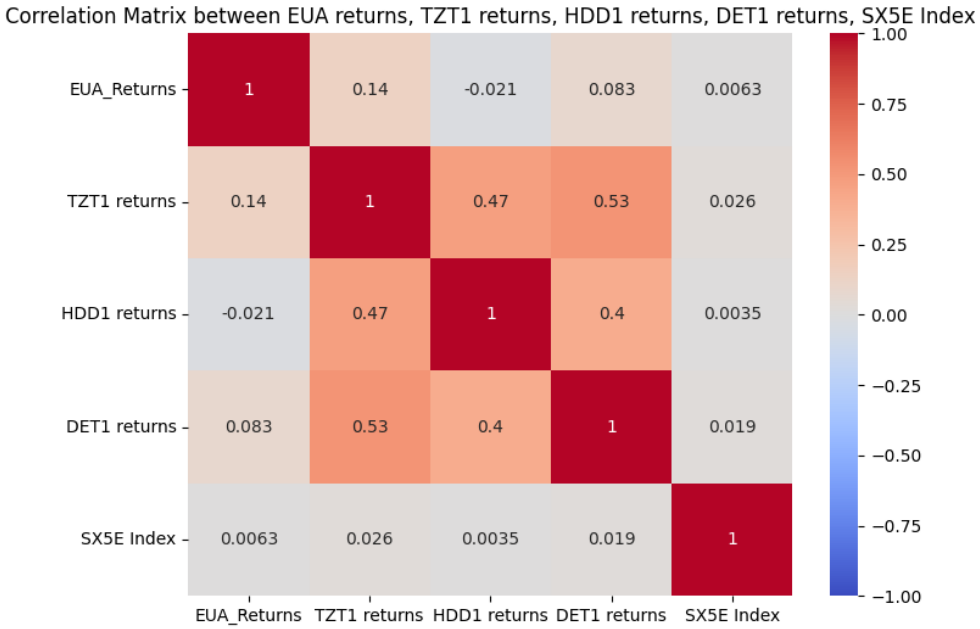

Figure 3.3: *Correlation Matrix for the 5-Day Logarithmic Returns and the SXSE Index.*

The correlation matrix shows generally weak relationships among the variables, with the strongest correlations observed between TZT1, HDD1, and DET1 returns. EUA returns exhibit only mild correlations with TZT1 and DET1 returns, and are nearly uncorrelated with HDD1 returns and the SX5E Index. The SX5E Index shows almost no correlation with any of the other variables.

4. Methodology

This Section describes in detail all the methodological steps that have been taken to build the automated trading system. These steps are the following:

- We start with a pre-processing of the data, in order to guarantee that our analysis can be implemented in an effective way.
- Section 4.3 describes the technical indicators used as features in the training of the algorithms, including the mathematical formulations and intuitions underlying their application. Only a selection of technical indicators will be considered, although many more could have been included. This selection is made on the basis of a review of the indicators most commonly used by practitioners and academics, taking into account previous studies on their predictive capacity.
- The implementation of the Random Forest and XGBoost algorithms will be explained in section 4.2. Since the entire experiment is performed using Python code, the library chosen to implement the algorithms and define their hyperparameters is scikit-learn. Each of the hyperparameters will be explained in depth and the whole training process will be described step by step.
- Finally, we will design a trading strategy based on the predictions obtained from the models defined previously.

4.1 Data Preprocessing

Data preprocessing is very important in this type of procedure as it ensures that the information used in an analysis or model is of high quality and well structured. This includes cleaning erroneous data and missing values, normalizing values to avoid biases, and combining data from different tables. Good preprocessing improves the accuracy of the models, reduces noise, and ensures more reliable and meaningful results.

We have implemented the following steps in the data preprocessing:

1. We have identified all the days on which these contracts (in addition to TZ1, HHD1 and DET1 contracts) were not traded and we have eliminated these rows. In other words, we have eliminated the missing values.
2. As we have already explained in subsection 1.2, a rolling technique has been performed to obtain a unique price series for the EUA futures, and we have computed weekly logarithmic returns.

3. We have created a unified table with all available data for both EUA futures and fundamental data. This table contains the prices and returns of the EUA, gas, coal, and electricity futures contracts. We have also added the Dark Spark Spread, Clean Spark Spread, and the EURO STOXX 50 index.

4.2 Decision Trees, Random Forest, and XGBoost

Machine Learning (ML) is a branch of artificial intelligence that allows machines to learn from data and past experiences in order to identify patterns and make decisions with minimal human intervention.

In our case, we will use supervised learning algorithms to build our trading strategy. Supervised learning consists of training a model using a set of labeled data, i.e. input data along with the correct responses. The goal is to learn to predict the correct output for new inputs based on the patterns identified during the training.

The data matrix used for training the model will consist of a DataFrame in which each record represents the weekly returns of EUA futures. This DataFrame will also include, as explanatory variables, the weekly returns of the fundamental factors described in section 3 (gas, coal, and electricity futures). The Dark and Clean Spark Spread values are also incorporated, as well as the values of the EURO STOXX 50 index.

To detect seasonality, we add a column indicating the month corresponding to each observation. We also include the values of the technical indicators that will be applied to the price of EUA Futures (simple moving average (SMA), the relative strength index (RSI), the MACD indicator, and the Bollinger Bands).

The final column of the DataFrame serves as the target variable, containing a categorical label that classifies the expected performance of EUA futures returns for the **following week** into one of five categories: Very Bad, Bad, Neutral, Good, or Very Good.

We will apply a percentile-based methodology to build these labels. First, we order from lowest to highest all the weekly returns available in the sample. Then, we compute the 20th, 40th, 60th, and 80th percentiles. From these values, we define the ranking intervals as follows:

- Return $<$ 20th percentile $\rightarrow$ Very bad.
- 20th percentile $\leq$ return $<$ 40th percentile $\rightarrow$ Bad.
- 40th percentile $\leq$ return $<$ 60th percentile $\rightarrow$ Neutral.
- 60th percentile $\leq$ return $<$ 80th percentile $\rightarrow$ Good.

- Return $\geq$ 80th percentile $\rightarrow$ Very good.

We will apply Random Forest and XGBoost as the supervised learning algorithms to train and make predictions. Our main objective is to develop models that can predict the performance category of EUA Futures returns for the following week. This prediction will allow us to design and evaluate a trading strategy based on the information provided by these models. We present now the theoretical development of both algorithms and their key features before proceeding to their practical implementation. We start by describing decision trees, the basis on which these algorithms are built.

4.2.1 Decision Trees

A decision tree is a type of ML algorithm used to predict the value or category of a variable (for example, whether a stock price will go up or down) based on rules it learns from a certain dataset. The decision tree applies, step by step, a series of rules that divide the data into smaller groups, and then makes a different prediction for each group.

In our case, since we want to classify the returns, we use what is called a classification tree. This type of tree predicts the probability of belonging to a certain category (in our case that the next week's performance will be very bad, bad, neutral, good, or very good), based on how many times each category appears in each group (relative frequency), or simply chooses the most common option within that group.

Each of the rules previously mentioned is based on a variable (e.g., last week's returns) and divides the data into two groups, those below a certain value and those above. This process forms what we call a binary tree, where

- The **root** is the starting point (containing all the data).
- The **nodes** represent the decision points defined by each rule.
- The **branches** connect those decision points.
- The **leaves** are the end points where the prediction is made.

Unlike other models, where you can directly see the effect of each variable on the outcome, the interpretation in a tree comes from following the path from root to leaf: each step shows how certain characteristics lead to a final decision. This makes trees able to capture more complex relationships between variables that other models do not detect as easily.

Figure 4.4 shows how the tree learns a rule. During the training, the algorithm examines all possible variables and tries different values to split the data into two parts. It chooses the best split: the one that separates the data in such a way that prediction errors are minimized as much as possible. In deciding this, it also considers how much data is in each

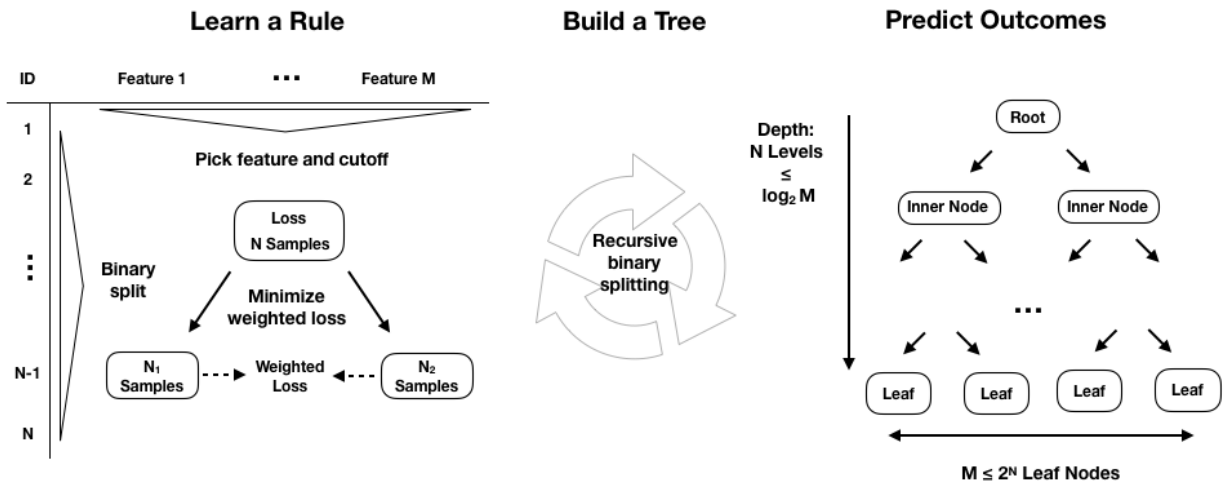

Figure 4.4: *How a decision tree learns rules from data. Source: [7].*

resulting group.

As mentioned above, a classification tree is a model that predicts a category from past data. To do so, this tree groups the data into different branches according to certain conditions, and in each final group (leaf node) it assigns the most common category among the examples that arrived there. This way of deciding is called mode. For example, if at a leaf node most of the examples were cases where an action went up, the tree will predict "go up" for new data arriving at that node.

In addition to predicting a category, the tree can also predict how likely the prediction will be true. For example, it can predict "that there is an 80% chance that it will go up and a 20% chance that it will go down," based on the proportion of cases that were in that node during the training.

We use a technique called recursive binary splitting to build the tree. Basically, the tree tries different ways of separating the data into two groups to improve the predictions. A simple option to assess the quality of the split is to look at the error rate, that is, how many examples in a group do not belong to the most common category. However, this measure is limited, and hence more effective measures are used, such as the Gini impurity or the entropy.

Both measures help to better evaluate how "pure" a group is. A group is pure if it contains instances of a single class (e.g., all the data in the group has the label "the stock will go up"). The purer a node is, the better it classifies that part of the data space. These measures help the tree to find divisions that separate the classes well and therefore make better predictions. In other words, they help to know whether the groups formed are "pure", that is, whether examples of a single category clearly predominate in each group. In sum,

both measures are used to evaluate how good a division is in a classification tree.

We calculate these two measures. Suppose that we have a classification problem with K possible categories: $0, 1, \cdots, K-1$ (e.g., $K = 2$ if it is a binary case). Let p_{mk} denote the proportion of samples at a node m of the tree that belong to the class k. Then, the measures to evaluate the purity of a group are calculated as follows:

- **Gini Impurity**:

$$\text{Gini} = \sum_{k=1}^{K} p_{mk}(1 - p_{mk})$$

- **Entropy**:

$$\text{Entropy} = -\sum_{k=1}^{K} p_{mk} \log(p_{mk})$$

Both measures reach their maximum value when the proportions of the class are balanced, i.e., when there are equal numbers of examples of each class (e.g., 50% of each if there are two classes). This indicates that the node is very impure. On the other hand, when most of the examples in the node belong to a single class (e.g., 90% or more), these measures decrease, indicating that the node is purer and thus better for prediction purposes. In summary, unlike simply counting how many errors would be made (the classification error rate), these measures penalize more heavily when the node is mixed and there is no clearly dominant class.

Figure 4.5 shows that both Gini impurity and entropy are maximized when class ratios are equal (0.5 in the binary case). Both measures decrease as the class proportions approach extreme values (zero or one). Finally, they imply a node impurity penalty greater than the classification error rate.

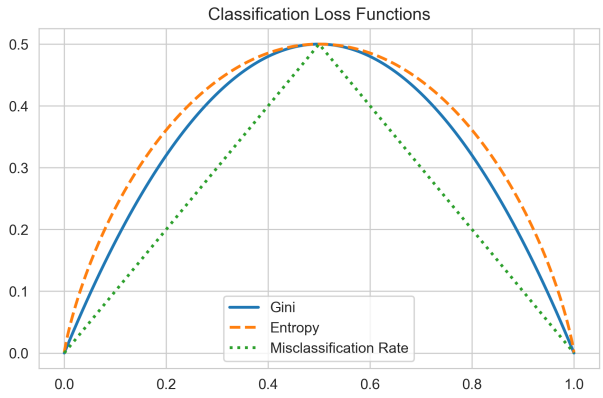

Figure 4.5: *Classification loss functions. Source: [7].*

Therefore, to decide how to make the splits (binary splits), the tree examines all the available variables and tests different cut-off values (e.g., in our case, RSI > 50), dividing the data into two groups. It then evaluates how good each split is using one of these two measures of impurity, Gini impurity or entropy. The idea is that the purer a group is, the better the split.

For each possible cut-off point, the tree calculates the total impurity as a weighted average between the impurity of the two resulting groups:

$$\text{Total impurity} = \frac{n_{\text{left}}}{n} \cdot \text{Impurity}_{\text{left}} + \frac{n_{\text{right}}}{n} \cdot \text{Impurity}_{\text{right}}$$

where n_{left} and n_{right} are, respectively, the numbers of observations at the left and right nodes and n is the total number of observations at the original node.

This process is repeated for each variable and each possible value, and the tree always chooses the split that generates the lowest total impurity. Thus, through this trial-and-error procedure, the tree grows, forming purer and purer nodes, which allows it to make more accurate predictions.

Figure 4.6 shows how a decision tree would look like once the process is completed.

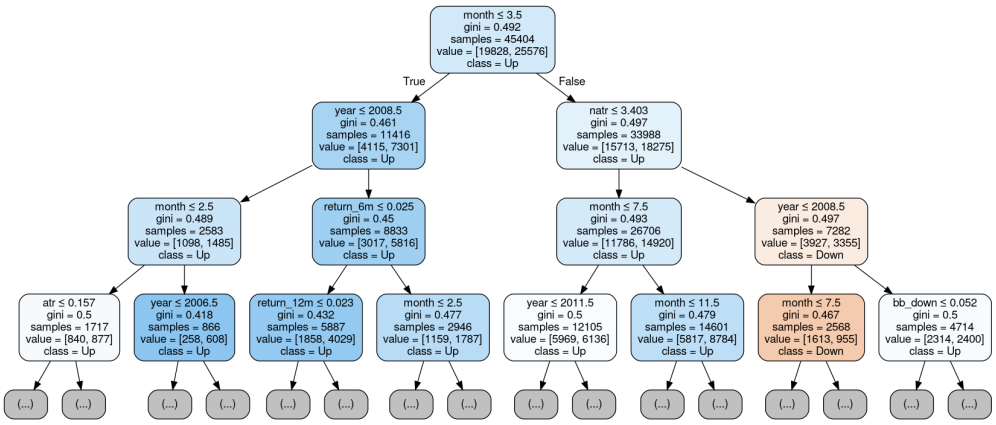

Figure 4.6: *Visualization of a classification tree. Source: [7].*

This figure shows a classification decision tree, used to predict whether an event will be **Up** or **Down**, as might be the case for market movements. Each node contains several key parameters:

- **Split condition** (e.g., `return` $\leq$ `0.025`): This is the rule that splits the data. If the condition is met, the path continues to the left branch; otherwise, to the right one.
- **Gini**: Gini impurity index. A value close to 0 indicates that the node contains mainly samples from a single class (a pure node), while values near 0.5 indicate a mix of classes.

- **Samples**: Total number of observations (rows) that have reached this node from the root of the tree.
- **Value** = **[**x**,** y**]**: Indicates how many observations of each class are present in the node. For example, `[19828, 25576]` indicates that 19,828 (resp., 25,576) observations belong to the `Down` (resp., `Up`) class.
- **Class**: The class predicted by the node, determined by a simple majority within `value`. This is the class assigned if a sample reaches this node.

This type of tree is traversed from the root (top) to the leaves (bottom), applying the conditions at each node to decide which branch to follow. As one moves downward, the nodes group increasingly similar observations, which improves the accuracy of the predictions.

Decision trees are not only valuable for their transparency and interpretability, but also serve as the foundation for more powerful models: ensemble learning methods. These approaches combine multiple individual models with the goal of improving predictive performance and reducing errors such as overfitting.

There are two main groups within ensemble learning, depending on how the base models are trained and how their predictions are combined:

- **Averaging methods**, such as *bagging* and *Random Forest*, train several estimators independently and then average their predictions. If the individual models are unbiased and their errors are not strongly correlated, averaging their outputs tends to reduce the variance and improves the stability of the final model. This logic resembles the diversification of a financial portfolio: combining uncorrelated assets reduces volatility without affecting the expected return.
- **Boosting methods**, such as *XGBoost*, train the base models sequentially. Each new model focuses on correcting the errors made by the previous ones, with the goal of reducing the overall bias of the combined model. The result is a stronger ensemble departing from weak models.

The **bagging** (short for *bootstrap aggregation*) mentioned previously is a procedure that generates multiple random subsets of the training set (with replacement) and trains an individual model, in this case, a decision tree, on each subset. Subsequently, the predictions of the different trees are combined (by majority voting) to obtain a more robust final prediction with a lower variance.

4.2.2 Random Forest

Random Forest is an extension of bagging applied to decision trees. In addition to generating random subsets of data for each tree, this technique introduces additional randomness

by randomly selecting a subset of features to evaluate at each split in the tree. This contributes to the generated trees being less correlated with each other, which further decreases the variance of the ensemble.

For classification problems, the number of variables selected at each node is usually the square root of the total number of features. This randomization in variable selection allows the trees to explore different combinations and structures, producing a more diverse and robust model.

Figure 4.7 illustrates the main features of the Random Forest algorithm.

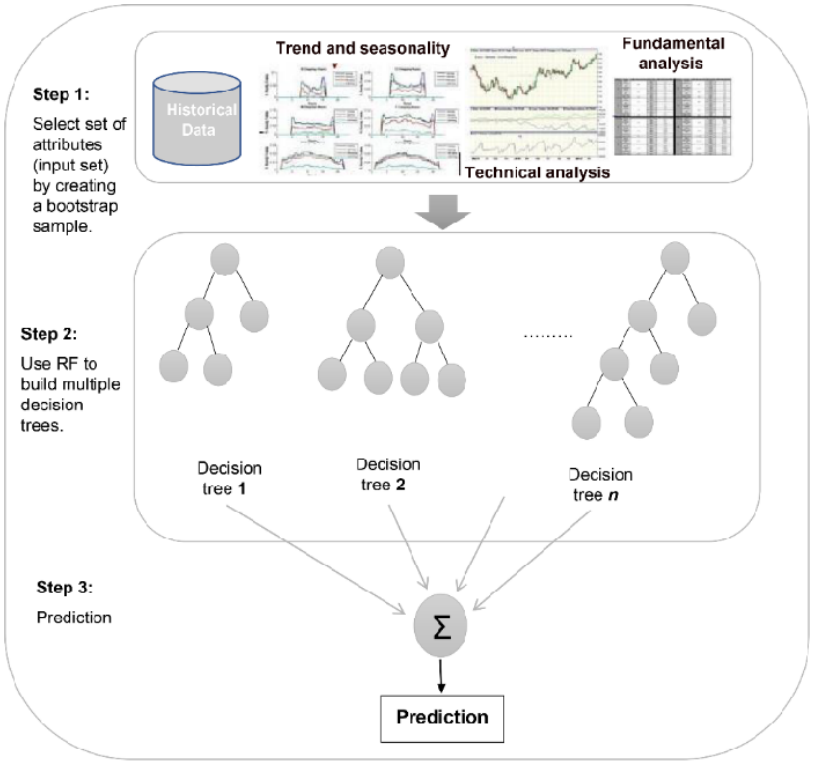

Figure 4.7: *Schematic diagram of the Random Forest algorithm. Source: [7].*

The process involves three steps:

1. Selecting a set of attributes from historical data, technical, and fundamental analysis
2. Next, multiple independent decision trees are constructed from bootstrap samples.
3. Finally, the individual predictions from each tree are combined (by voting or averaging) to generate a more robust final prediction.

Hyperparameter Fitting and Training

In ML models, it is important to distinguish between parameters and hyperparameters. Parameters are internal values that the model learns automatically from the training data. Common examples are the coefficients in a linear regression or the weights in a neural network. On the other hand, hyperparameters are settings external to the model that are defined prior to the training process and are not learned directly from the data. These hyperparameters influence how the internal parameters are tuned. For example, the number of trees in a random forest or the maximum depth of each tree are examples of hyperparameters.

Among the most important hyperparameters for the training of a Random Forest, we can highlight the following:

- **n_estimators**: number of trees in the forest. A higher value typically leads to a better performance, although it increases the computational cost.
- **max_features**: maximum number of features considered at each split. Lower values reduce the correlation between trees but may increase bias.
- **bootstrap**: indicates whether sampling with replacement is used (default is `True`).
- **oob_score**: enables internal validation using out-of-bag samples.
- **max_depth** and **min_samples_split**: they control, respectively, the depth of the trees and the minimum number of samples required to split a node.
- **n_jobs**: number of CPU cores used in parallel for training; `-1` uses all the available cores.

To adjust the hyperparameters of a ML model, the cross-validation technique is often used. This procedure consists of dividing the training data set into several subsets or "folds". The model is successively trained on one part of the data and evaluated on the remaining subset, rotating these divisions to cover the whole set. In this way, a more reliable estimate of model performance is obtained by avoiding over-dependence of the results on a single partition of the data. Cross-validation allows to compare different combinations of hyperparameters and select those that provide the best balance between predictive ability and generalization.

The following pseudocode illustrates how the Random Forest algorithm works.

Algorithm 1 Random Forest Algorithm (Classification)

Require: Training dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, number of trees T, number of features per split m

Ensure: Final classifier using majority voting

for $t = 1$ **to** T **do**
  Generate a bootstrap sample D_t from the original dataset D
  Train a decision tree h_t using D_t:
  for each node in the tree **do**
    Randomly select m features without replacement
    Determine the best split among the m features
    Split the node based on the best split
  end for
end for
Classifying a new instance x:
Get predictions from all trees: $h_1(x), h_2(x), \ldots, h_T(x)$
Return the class with the majority vote among all predictions

4.2.3 XGBoost

We start explaining Gradient Boosting, a supervised learning technique based on the sequential ensemble of weak models, typically decision trees. Its main objective is to build a robust model by iteratively adding new trees that correct the errors made by the previous set, minimizing a certain differentiable loss function.

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be the training data set, where $x_i \in \mathbb{R}^d$ and y_i are, respectively, the predictor and the objective variables. The model is built in an additive way:

$$F_T(x) = \sum_{t=1}^{T} \eta \cdot h_t(x)$$

where

- $F_T(x)$ is the prediction of the final model after T iterations,
- $\eta \in (0, 1]$ is the learning rate that regulates the contribution of each tree,
- $h_t(x)$ is the decision tree trained at iteration $t = 1, 2, \cdots, T$.

At each iteration t, the objective of the algorithm is to improve the current model $F_{t-1}(x)$. For this purpose, a new model $h_t(x)$ is trained, which does not attempt to directly predict the target value y_i, but to correct the errors of the current model.

For this purpose, we compute the **negative gradient** of the loss function[5] $L(y_i, F(x_i))$ with respect to the current model prediction. That is, for each observation i, we calculate

$$g_i^{(t)} = - \left. \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x_i)=F_{t-1}(x_i)}$$

This value $g_i^{(t)}$ indicates the direction and the magnitude in which the prediction $F(x_i)$ should be adjusted to reduce the error. If the value is positive (resp., negative), it means that the current prediction is too low (resp., high). Therefore, the new model $h_t(x)$ is trained to predict these values, i.e. to learn to correct the errors of the previous model.

When using a method (such as *XGBoost*) that incorporates second-order optimization, we also calculate the **Hessian** of the loss function, that is, its second derivative with respect to the prediction. This allows a more accurate update of the model. The hessian for each observation i is defined as:

$$H_i^{(t)} = \left. \frac{\partial^2 L(y_i, F(x_i))}{\partial F(x_i)^2} \right|_{F(x_i)=F_{t-1}(x_i)}$$

This value represents the curvature of the loss function and allows us to adjust the contribution of each correction more precisely.

The new tree is trained using both gradients and Hessians, and its leaves are fitted by minimizing a regularized loss function. Finally, the model is updated as:

$$F_t(x) = F_{t-1}(x) + \eta \cdot h_t(x)$$

This procedure is repeated until a predetermined number of iterations is reached or until the improvement in the loss function stabilizes.

Having understood how *Gradient Boosting* works, it is possible to introduce now **XGBoost** (*Extreme Gradient Boosting*), one of its most efficient and widely used implementations in practice.

XGBoost is based on the same principles as traditional Gradient Boosting, but introduces several improvements that optimize both the computational performance and the predictive capability of the model. Its main contributions include the following:

- The use of two regularization techniques[6] (L_1 and L_2) to control the complexity of the trees and to reduce the risk of overfitting.

[5] One of the most commonly used loss functions is the *log loss* function, defined as $L(y_i, F(x_i)) = -\left(y_i \log F(x_i) + (1 - y_i) \log(1 - F(x_i))\right)$.

[6] The L_1 regularization (Lasso) adds a penalty proportional to the absolute values of the coefficients, encouraging sparsity: $\Omega(f) = \lambda \sum_j |w_j|$ where λ controls the regularization strength and w_j represents the leaf weights. The L_2 regularization (Ridge) penalizes the squared magnitude of the leaf weights, leading to smoother models: $\Omega(f) = \lambda \sum_j w_j^2$.

- The use of parallel computing to significantly speed up the model building.
- A more efficient tree construction algorithm, which allows faster split evaluations.
- The possibility of using both observation and feature subsampling, improving the model robustness and reducing its variance.
- An internal error evaluation system using *out-of-core computation* for working with data sets that do not fit in memory.

As in other ML models, in XGBoost it is essential to adjust its hyperparameters to achieve a proper balance between bias and variance. Among the most important hyperparameters we can indicate the following ones:

- **n_estimators**: total number of trees to be built sequentially.
- **learning_rate** (η): learning rate that controls the contribution of each tree to the final model. Lower values imply a larger number of trees but better generalization.
- **max_depth**: maximum depth of each tree. Deeper trees capture more complex relationships, but increase the risk of overfitting.
- **subsample**: fraction of the training set used to build each tree. It helps prevent overfitting.
- **colsample_bytree**: proportion of variables randomly selected to build each tree.
- **reg_alpha** and **reg_lambda**: L_1 and L_2 regularization coefficients that penalize the complexity of the model.

As in the case of Random Forest, the most common technique for optimizing these hyperparameters is cross-validation, which allows selecting the configuration that offers the best performance on unused data.

The following pseudocode illustrates how the *XGBoost* algorithm works.

Algorithm 2 XGBoost Algorithm (Classification)

Require: Training dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, number of boosting rounds T, learning rate η
Ensure: Final model $F(x)$ as an additive ensemble of weak learners

```
Initialize model with a constant prediction: F_0(x) = arg min_c Σ_{i=1}^{N} L(y_i, c)
for t = 1 to T do
    Compute gradients (first derivatives): g_i = ∂L(y_i, F_{t-1}(x_i)) / ∂F_{t-1}(x_i)
    Compute hessians (second derivatives): H_i = ∂²L(y_i, F_{t-1}(x_i)) / ∂F_{t-1}(x_i)²
    Fit a regression tree h_t(x) to the training data {(x_i, g_i, H_i)}
    Compute optimal leaf weights for h_t(x) by minimizing regularized loss
    Update model: F_t(x) = F_{t-1}(x) + η · h_t(x)
end for
Predict: Return F_T(x)
```

Appendix A includes a step-by-step derivation of the objective function, that is, the function that the XGBoost algorithm aims to minimize.

4.3 Technical Indicators

Technical indicators are tools used by traders and practitioners to analyze price behavior in financial markets. These indicators are based on mathematical formulas that process data such as price, volume, and time. They are then displayed on charts to help make decisions about timing and dynamics of trading (purchases and sales) of financial assets.

In [12] it is emphasized that technical indicators should not be used in isolation, but in conjunction with price behavior analysis and other tools. Their main usefulness is to confirm trends, point out possible market turns, and help the trader make more objective decisions.

Any technical analysis is based on three fundamental premises:

1. The market discounts everything, that is, all relevant information is already reflected in the prices.
2. Prices move in trends, and a trend is more likely to continue than to reverse immediately.
3. History repeats itself, as market behavior patterns tend to repeat themselves over time.

Technical indicators are usually divided into two groups: trend-following indicators (such as moving averages) and oscillators (such as RSI or MACD), which show whether an asset is overbought or oversold.

For our study, we will calculate a series of technical indicators for the price of EUA futures and add them to the unified table mentioned above. ML algorithms will be applied to this table to predict the returns of these futures. As we will see below, some technical indicators are quite relevant in predicting these returns.

In this section we will explain in detail the technical indicators we have used and will provide a theoretical background for them. We have developed on our own all the examples used in this section. For these examples we have used data other than EUA futures returns as in this way the examples are more illustrative. To generate the data, we simulated a price series by cumulatively summing normally distributed random values and adding a constant to introduce a trend.

We have used the following technical indicators: SMA 20 AND 50 (Simple Moving Average of 20 and 50 periods), RSI (Relative Strength Index), MACD (Moving Average Convergence Divergence), and the Bollinger Bands. These indicators were selected because they effectively represent both trend-following tools and oscillators and are among the most widely used by traders in financial markets.

4.3.1 Simple Moving Average (SMA)

We start by defining a moving average. A simple moving average (SMA) is an indicator that calculates the average of the prices of an asset (in our case, EUA futures) over a specific period. It is computed by summing the closing prices of an asset over a period and dividing this sum by the number of days in that period. For example, a 20-day SMA is the average of the closing prices of the last 20 days. Its mathematical expression is

$$\mathrm{SMA}_{n,t} = \frac{1}{n} \sum_{i=0}^{n-1} P_{t-i}$$

where:

- $\mathrm{SMA}_{n,t}$ is the simple moving average of n periods at time t,
- P_{t-i} is the closing price of the asset on the day $t - i$,
- n is the number of days in the period considered.

The SMA is used to identify the general price trend, as it smooths volatility and allows visualizing the market direction. It is also useful for generating buy and sell signals. For example, when the price crosses above (resp., below) the SMA it can be interpreted as a buy (resp., sell) signal. Its use is due to its simplicity, as it is easy to calculate and to interpret.

Figure 4.8 provides an example for a SMA for 20 days. We can see that the price crosses below the SMA on the 40th day, which could mean the beginning of a downtrend. After

that crossing, the price effectively fell. Then, the price crosses above the SMA on the 85th day, and from that point on, it started to rise. However, it is important to be cautious since, for example, on the 30th day the price also crossed below the SMA, but although there was a small drop, it was followed by a significant rise in prices. This shows that crosses of the SMA are not always clear signals, and it is necessary to consider other factors before making decisions.

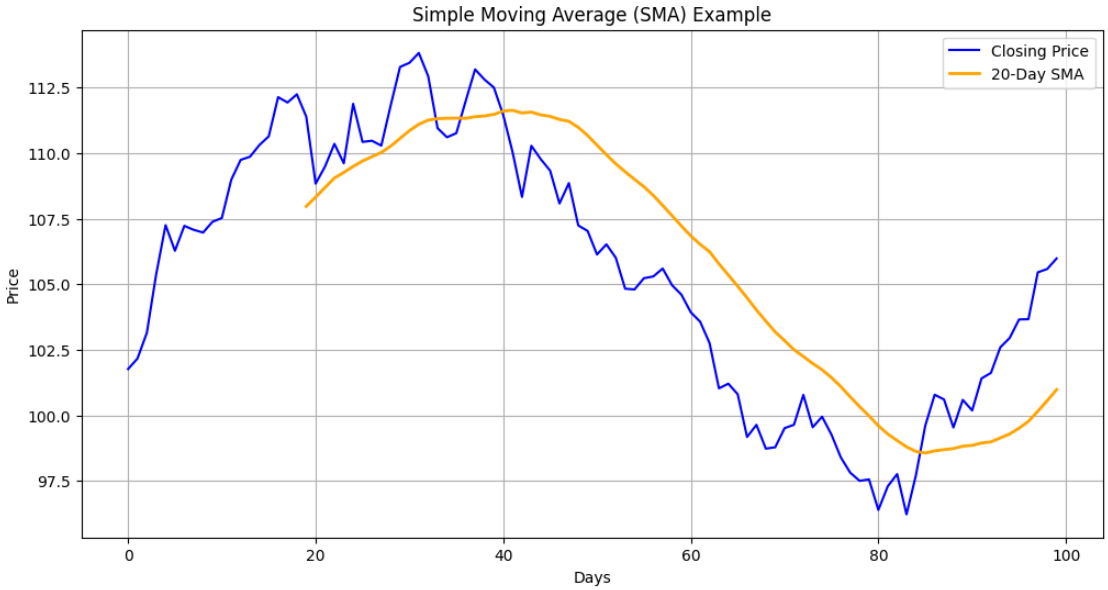

Figure 4.8: *Simple Moving Average for 20 days.*

4.3.2 Relative Strength Index (RSI)

This indicator was introduced and developed in [13]. It measures the speed and change of price movements of an asset over a specified period. The RSI ranges from 0 to 100 and is used to identify whether an asset is overbought or oversold. Generally, a value above 70 (resp., below 30) is considered overbought (resp., oversold).

An asset is overbought when its price has risen too much and too fast. This asset may be overvalued and there could be a drop in price in the near future. Similarly, an asset is oversold when its price has fallen too much and too fast. This indicates that it may be undervalued and there could be a rise in price in the near future. The RSI is computed as follows:

$$RSI = 100 - \frac{100}{1 + RS} \tag{4.4}$$

where

$$RS = \frac{\text{Average profit for the period}}{\text{Average loss for the period}}$$

The average profit is calculated by summing all gains during the period and dividing by the number of periods considered, while the average loss is obtained by summing all losses (expressed as positive values) over the same period and dividing by the number of periods. If the asset has had more days with gains than losses, the RS value will be higher. If the asset has had more (resp., less) days with gains than losses, the RS value will be higher (resp., lower). The RSI is intended to express relative strength on a scale from 0 to 100. Its calculation involves the following steps:

1. For each period, we compute the gain (resp., loss) if the price has gone up (resp., down).
2. We calculate the average profit and the average loss during the period.
3. We apply equation (4.4) to obtain the RSI.

Figure 4.9 illustrates the RSI for 14 days. It can be seen that, between days 15 and 20, the RSI is above 70%, which could indicate that the asset is overbought. As a result, 20 days later, the price went down. Similarly, between days 50 and 70, the RSI is below 30%, suggesting that the asset is oversold. As we can see, from day 80 on, the price started to rise.

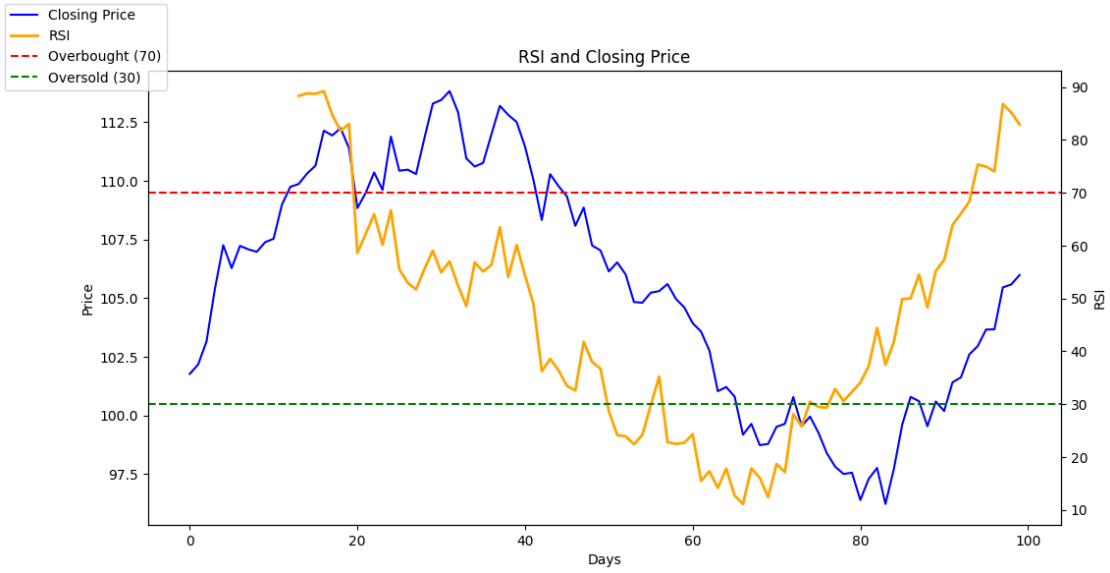

Figure 4.9: *Relative Strength Index for 14 days.*

4.3.3 Moving Average Convergence Divergence (MACD)

This indicator measures the difference between two exponential moving averages (EMAs) of different periods (usually 12 and 26 days). It seeks to identify changes in the direction,

strength, and duration of a trend in the price of an asset.

Unlike simple moving averages (SMAs) which give equal weight to all prices, exponential moving averages (EMAs) give more weight to the most recent prices when calculating the average. Exponential moving averages are calculated as follows.

$$\text{EMA}_{n,t} = \alpha \cdot P_t + (1 - \alpha) \cdot \text{EMA}_{n,t-1}$$

where

- $\text{EMA}_{n,j}$ is the exponential moving average of n periods at time $j = t - 1$, t,
- P_t is the closing price at time t,
- $\alpha = \frac{2}{n+1}$ is the smoothing factor,
- n is the number of periods.

The MACD is defined as follows:

$$\text{MACD}_t = \text{EMA}_{12,t} - \text{EMA}_{26,t}$$

where

- MACD_t is the value of the MACD on the day t,
- For $i = 12, 26$, $\text{EMA}_{i,t}$ is the exponential moving average of i periods at time t,

Next, we compute the Signal Line, which is an exponential moving average (usually of 9 periods) of the MACD itself:

$$\text{Signal}_t = \text{EMA}_{9,t}(\text{MACD}_t)$$

The interpretation of the MACD is based on the crossover between the MACD and the Signal Line. We can consider two alternatives:

1. When the MACD crosses above the Signal Line, it is interpreted as a buy signal. This indicates that the upward momentum is gaining strength, as the fast moving average (12-day EMA) is rising faster than the slow moving average (26-day EMA). In other words, recent prices are rising faster.
2. On the other hand, if the MACD crosses below the Signal Line, it is considered a sell signal. In this case, the positive momentum weakens and the MACD falls below its average. This may indicate that the downtrend is increasing, which could anticipate a drop in price and the start of a downtrend.

This indicator is very popular because it combines two important things: trend following and the strength of the movement (momentum). This makes it useful for understanding not only where the price is going, but also how strongly it is moving.

Figure 4.10 shows the time evolution of this indicator. We observe that, around day 20, the MACD (orange line) crosses above the Signal Line (red line), which can be interpreted as a buy signal. From that point, the price starts to rise, indicating that the bullish momentum gained strength in the following days. Later, near the 70th day, the MACD crosses below the Signal Line, indicating a sell signal, as it suggests bearish momentum. After this crossover, the price starts to move lower in the following days. This reflects how the MACD can anticipate possible changes in market direction.

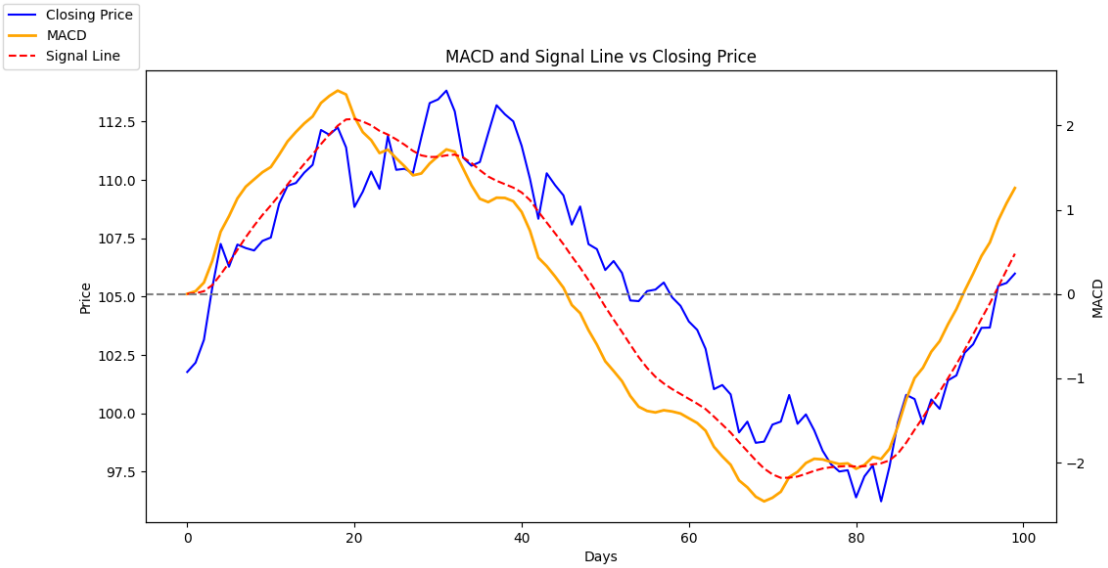

Figure 4.10: *Moving Average Convergence Divergence.*

4.3.4 Bollinger Bands

These bands are a technical indicator designed by John Bollinger. This indicator consists of an upper band, a lower band, and a simple moving average (SMA) in the middle. This indicator measures the volatility of the asset and detects possible overbought or oversold levels. The bands are adjusted according to the standard deviation of the price. This allows them to expand or contract according to the volatility of the asset. The higher the volatility, the wider the bands, and vice versa.

The mathematical expressions for these bands are the following:

$$\begin{aligned}\text{Middle Band}_{n,t} &= \text{SMA}_{n,t} \\ \text{Upper Band}_{n,t} &= \text{SMA}_{n,t} + k \cdot \sigma_{n,t} \\ \text{Lower Band}_{n,t} &= \text{SMA}_{n,t} - k \cdot \sigma_{n,t}\end{aligned}$$

where

- $\text{SMA}_{n,t}$ is the simple moving average of n periods at time t,
- $\sigma_{n,t}$ is the standard deviation of prices over n periods at time t,
- k is the number of standard deviations ($k = 2$ is usually used),
- n is the number of days in the period considered.

When the price touches or rises above the upper band (resp., below the lower band), it can be interpreted as an overbought (resp., oversold) condition. The logic behind this interpretation is based on the principle of reversion to the mean. That is, if the price deviates greatly from its average (the middle band), it is expected that it will eventually tend to move back toward it. Therefore, when the price touches or exceeds one of the bands, it is interpreted as a possible reversal point or market adjustment.

Figure 4.11 shows that around the 30th day, the price touches the upper band (green line), indicating that the asset is overbought. After this crossover, the price starts to decline. Later, on day 65, the price touches the lower band (red line) indicating that the asset is oversold. After this contact, albeit gradually, the price starts to recover. Moreover, around day 80, the price is again very close to the lower band. After this approach, the price rises strongly.

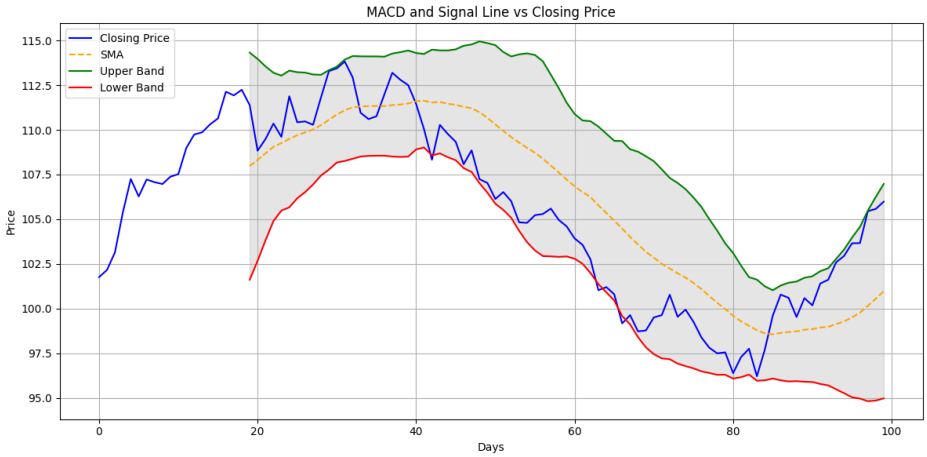

Figure 4.11: *Bollinger Bands.*

4.4 Trading Strategy

This subsection describes the design of a trading strategy based on ML models. In particular, we will use the predictions of the models described in the previous section to develop a trading strategy.

Each week, data corresponding to the current market situation is fed into the models. These data include the current returns of the EUA futures, the technical analysis indicators calculated in Technical Indicators, and the fundamental variables data described in Fundamental Variables. As mentioned above, the models provide a categorical prediction of what the next week's returns will be like, classifying them into one of five possible categories (Very Good, Good, Neutral, Bad, Very Bad).

Based on this prediction, a decision rule is established that defines the position to be taken in the market during that week:

- If the model predicts "very good" or "good" returns, a **long position** in the market is taken and the full weekly return is assumed.
- If the prediction is "neutral", **no position** is taken that week.
- If the prediction is "bad" or "very bad", a **short position** is taken and the weekly performance is assumed in the opposite direction.

This trading logic is applied for both the **Random Forest** and **XGBoost** models, generating two individual strategies.

Additionally, we define a mixed strategy that takes decisions in a more prudent way: a position (long, short, or neutral) is only taken if both models agree on the same signal for the week. In other words, the mixed strategy acts only when there is consensus between Random Forest and XGBoost, which is expected to reduce noise and increase signal robustness.

In order to simulate these strategies (and compare them with other more classical strategies as we will see in the next section), the last 400 days of the original data have been reserved to apply these strategies and evaluate the results. This data set will be called backtest data, and it is not used at any time for training or evaluation of the models.

In all the cases, we start with a fixed initial capital and calculate the time evolution of the capital according to the decisions generated by each strategy.

5. Results

This Section is divided into two parts. The first part evaluates the predictive performance of the Random Forest and XGBoost models using metrics such as the confusion matrix, as well as training, validation, and test errors. In addition, the importance of the variables is analyzed to identify those that have had the greatest relevance in the model's decisions.

The second part evaluates the financial performance of the trading strategies, comparing them with simpler approaches such as "Buy and Hold", technical analysis, and a random strategy. For this purpose, we present the time evolution of the capital during the backtest and a table with metrics such as return on investment (ROI), volatility, Sharpe ratio, and *max drawdown*.

5.1 Model Performance

During model training, data are divided into a training set and a test set.[7] The training set is used to adjust the model parameters and learn the patterns that allow predictions to be made. The test set, which has not been used in the training process, is used to evaluate the performance of the model on new data. This subsection presents the results obtained by applying the model to the test set using different evaluation metrics.

Before showing a table with the values of the different types of error, it is important to briefly explain the different types of error that can arise:

1. The **training error** indicates how well the model fits the data with which it was trained. A low value may mean that the model has correctly learned these patterns, although it does not imply that will generalize adequately to new data.

2. The **validation error** is calculated using data that the model has not seen during training, but which are used to fit its parameters. We obtained it by cross-validation and provided a more realistic estimate of its generalizability.

3. Finally, the **test error** is calculated on a completely independent data set, which has been used neither for training nor for validating the model. It is the best indicator of expected performance in real situations.

Both validation and test errors serve as estimators of the **generalization error**, that is, the model's ability to predict correctly on new and unseen data.

[7] Test data and backtest data should not be confused. Test data are part of the training process and is used to evaluate the model, while the backtest data are only used for the trading strategy.

The following table summarizes the results of the evaluation of model errors.

Model	Training Error	Validation Error (10-fold)	Test Error
Random Forest	0.0000	0.5281	0.5179
XGBoost	0.0000	0.5476	0.5309

Table 5.1: *Evaluation errors for Random Forest and XGBoost.*

Both models show zero training error, indicating a perfect fit to the training data. However, the validation and test errors show some overfitting and similar generalizability in both cases.

We present now the confusion matrices corresponding to each model. These matrices allow us to analyze the performance of the classification models, showing how the predictions are distributed against the true classes. Each row (resp., column) represents the true (resp., predicted) class.

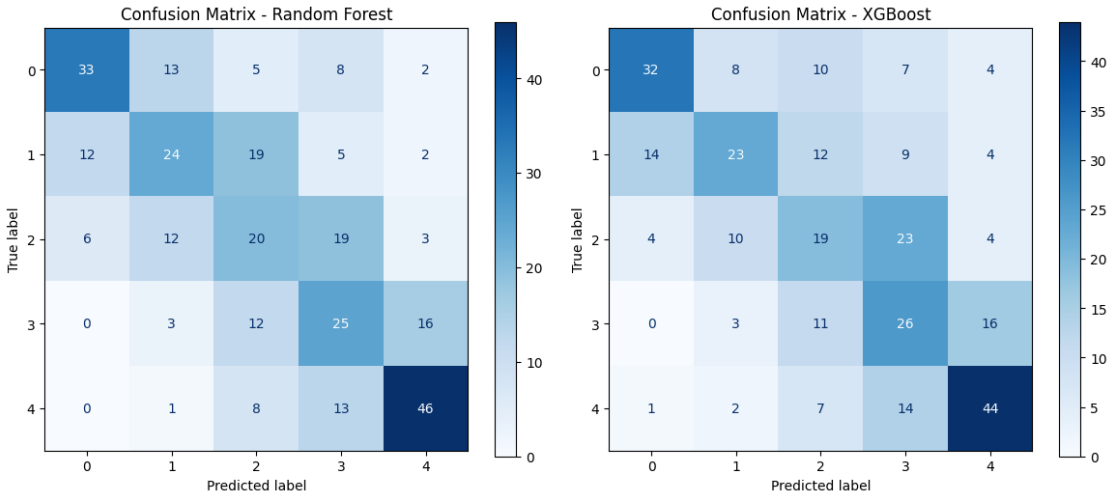

Figure 5.12: *Confusion matrices for Random Forest and XGBoost models.*

Both models tend to make errors in classes close to the true class. For example, when the model predicts "Very Good" performance and is wrong, the error usually falls into "Good" or "Neutral", and rarely into 'Bad' or "Very Bad". This decreasing pattern in errors with distance between classes is consistent across all rows, indicating that, although the prediction is not always accurate, the model maintains a consistent logic and its errors are not random or extreme.

Finally, both models allow us to analyze the importance of the variables (features), which allows us to identify the factors with the greatest influence on their prediction decisions. Fig-

ure 5.13 shows the most relevant variables for Random Forest and XGBoost in comparative form.

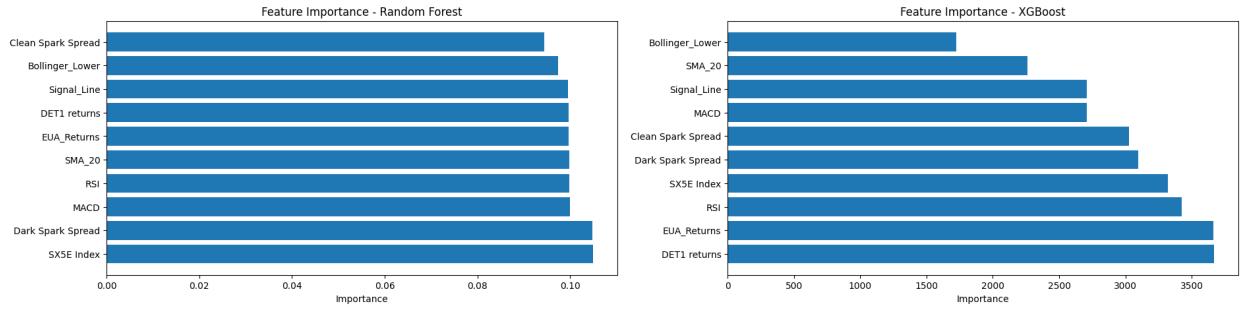

Figure 5.13: *Importance of variables according to each model.*

In Random Forest, the most influential variables are the SX5E index, the Dark Spark Spread, and the MACD. We can also see that the model attributes similar importance to all variables. On the other hand, XGBoost gives greater importance to the DET1 returns, the EUA futures returns and the RSI. This model gives more importance to certain variables instead of evenly distributing it like the previous one. This indicates that there are differences in the patterns that each model prioritizes in making its predictions.

5.2 Trading Performance

Finally, we present the performance of trading strategies derived from the predictions of Random Forest, XGBoost, and a combined approach that only trades when both models agree. These are benchmarked against three widely used references: “Buy and Hold", a technical indicators strategy, and a random strategy.

We explain briefly each strategy:

- **Buy and Hold**: It consists of maintaining a constant long position in the market throughout the entire analysis period, without making adjustments based on market conditions.
- **Technical Indicators Strategy**: It uses buy and sell signals generated by two technical indicators: the Simple Moving Average (SMA) and the Bollinger Bands. Positions are taken when relevant crossovers are detected in any of these indicators. We tested several combinations of technical indicators and chose this one as it delivered the best results.
- **Random Strategy**: It generates buy, sell, or no-entry signals randomly, serving as a benchmark strategy to assess whether other strategies provide real value beyond chance. To ensure a robust evaluation, we have generated 1000 random strategies,

each producing different trading outcomes. As a representative sample, we have highlighted the strategies associated with the 20th percentile (Random_1) and the 80th percentile (Random_2) based on the final capital obtained. These two strategies provide insight into the lower and upper performance ranges within the random set, helping to contextualize the effectiveness of other trading approaches.

Figure 5.14 shows the evolution of accumulated capital throughout the backtesting period for the different trading strategies analyzed.

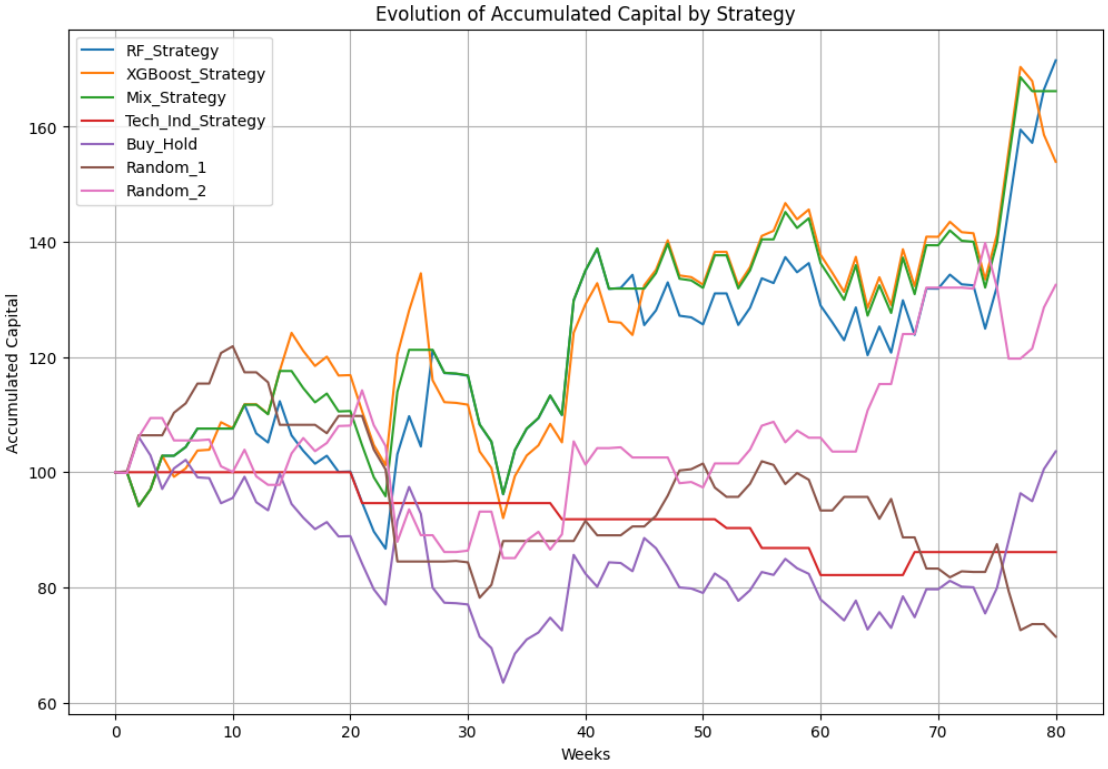

Figure 5.14: *Evolution of accumulated capital by trading strategy.*

The strategies based on ML models (Random Forest, XGBoost, and the mixed strategy) achieved a clearly superior capital growth compared to benchmark strategies.

The “Buy and Hold" strategy exhibits a more volatile performance, with the Random_1 strategy showing a clearly declining trajectory. On the other hand, the strategy based on technical indicators manages to preserve the initial capital but barely increases the capital over time. In contrast, the strategy Random_2 has performed reasonably well, achieving positive returns, but it has not reached the level of success seen in ML-based strategies.

The returns and risk profiles of the different strategies are evaluated and compared using financial performance metrics such as return on investment (ROI), volatility, Sharpe ratio, and maximum drawdown.

Strategy	ROI	Volatility	Sharpe Ratio	Max Drawdown
RF_Strategy	0.7153	0.0522	0.1309	-0.2280
XGBoost_Strategy	0.5392	0.0525	0.1039	-0.3160
Mix_Strategy	0.6618	0.0473	0.1360	-0.2065
Tech_Ind_Strategy	-0.1387	0.0118	-0.1605	-0.1786
Buy_Hold	0.0362	0.0531	0.0088	-0.4021
Random_1	-0.2856	0.0381	-0.1122	-0.4138
Random_2	0.3254	0.0442	0.0811	-0.2550

Table 5.2: *Trading strategy evaluation metrics (400 days),*

This table shows clearly that strategies based on ML models (Random Forest, XGBoost, and the mixed strategy) outperform the rest in terms of return (ROI) and Sharpe ratio and also provide a more efficient risk control (lower drawdowns and good return/volatility ratio). On the other hand, strategies such as the Random_1 strategy or the one based on technical indicators significantly underperform.

Table 5.3 presents the previously mentioned metrics annualized.

Strategy	ROI (Y)	Volatility (Y)	Sharpe Ratio (Y)	Max Drawdown
RF_Strategy	0.4049	0.0414	0.1039	-0.2280
XGBoost_Strategy	0.3122	0.0417	0.0825	-0.3160
Mix_Strategy	0.3771	0.0375	0.1079	-0.2065
Tech_Ind_Strategy	-0.0898	0.0093	-0.1274	-0.1786
Buy_Hold	0.0226	0.0421	0.0070	-0.4021
Random_1	-0.1909	0.0302	-0.0890	-0.4138
Random_2	0.1942	0.0351	0.0643	-0.2550

Table 5.3: *Annualized trading strategy evaluation metrics.*

6. Conclusion

This master's thesis research has shown that the use of ML algorithms combined with technical and fundamental data can be a useful tool for emissions trading decision-making. In this study we have found that techniques such as Random Forest and XGBoost can identify patterns in the behavior of EUA features and produce trading signals that resulted in trading strategies that performed better than conventional techniques.

Both models performed similarly on the validation and test sets, even though they had a perfect fit on the training set. The importance of both technical indicators and fundamental variables was highlighted by the feature importance analysis. The impact of the EURO STOXX 50 index, the spreads on electricity generation, and the returns on German electricity futures were among the key factors. Similarly, among all the technical indicators, the MACD was important in identifying market trends.

The construction of a mixed strategy based on model consensus reduced volatility and improved risk control—key factors in uncertain environments like carbon markets. Comparing it with conventional strategies (such as "Buy and Hold", those based solely on technical analysis, or even random approaches) reinforces the idea that ML techniques can make a difference in the profitability and efficiency of trading strategies.

Finally, this thesis has made a significant contribution to the literature on artificial intelligence in emissions markets. It has established the groundwork for future research that uses the same methodology in other markets or tests different ML algorithms. This opens new lines of investigation that compare various models and examine their use in diverse financial contexts. Such work could lead to more sophisticated and adaptive investment strategies.

Bibliography

[1] M. para la Transición Ecológica y el Reto Demográfico, "¿Qué es el comercio de derechos de emisión?" https://www.miteco.gob.es/es/cambio-climatico/temas/comercio-de-derechos-de-emision/que-es-el-comercio-de-derechos-de-emision.html, 2024.

[2] J.-P. Voß, "Innovation processes in governance: the development of 'emissions trading' as a new policy instrument," *Science and Public Policy*, vol. 34, no. 5, pp. 329–343, 2007. [Online]. Available: http://www.ingentaconnect.com/content/beech/spp

[3] P. Europeo, "Reducir las emisiones de carbono: objetivos y políticas de la ue," http://www.europarl.europa.eu, 2024.

[4] R. Roberts and C. Staples, "Emissions trading in the european union," *Capital Markets Law Journal*, vol. 3, no. 1, p. 5, 2008.

[5] J. H. v. Binsbergen, X. Han, and A. Lopez-Lira, "Man versus machine learning: The term structure of earnings expectations and conditional biases," *Review of Financial Studies*, vol. 36, pp. 2361–2396, 2023.

[6] Y. Zhang, Y. Zhu, and J. T. Linnainmaa, "Man versus machine learning revisited," https://ssrn.com/abstract=4899584, 2024, tuck School of Business Working Paper, forthcoming.

[7] S. Jansen, *Machine Learning for Algorithmic Trading*, 2nd ed. Birmingham, UK: Packt Publishing, 2020. [Online]. Available: https://www.packtpub.com/product/machine-learning-for-algorithmic-trading-second-edition/9781839217715

[8] J. Villamizar Díaz, "A deep learning approach to forecasting eua future contracts," Master's thesis, Universidad de los Andes, 2020, disponible en: http://hdl.handle.net/1992/51454.

[9] M. Bruttocao, "Automated stock trading system based on random forest algorithm: An application to the italian utilities sector," Master's thesis, Ca' Foscari University of Venice, 2023, https://hdl.handle.net/20.500.14247/6012.

[10] M. Aloud, "Designing strategies for autonomous stock trading agents using a random forest approach," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 12, no. 7, 2021, department of Management Information System, College of Business Administration, King Saud University, Saudi Arabia. [Online]. Available: https://thesai.org/Downloads/Volume12No7/Paper_11-Designing_Strategies_for_Autonomous_Stock_Trading.pdf

[11] S. G. C. I. Platts, "Specifications guide: European electricity," Available from S&P Global Commodity Insights, October 2024, latest update.

[12] J. J. Murphy, *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. New York, USA: New York Institute of Finance, 1999. [Online]. Available: https://sharemarketclasses.in/wp-content/uploads/2024/08/Technical_Analysis_of_the_Financial.pdf

[13] J. W. J. Wilder, *New Concepts in Technical Trading Systems*. Greensboro, North Carolina: Trend Research, 1978.

[14] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, 2016, pp. 785–794. [Online]. Available: https://arxiv.org/abs/1603.02754

Appendix A

Derivation of the objective function in XGBoost

The optimization problem to be solved in the XGBoost algorithm is as follows:

$$h_t = \arg\min_{h \in \Phi} \sum_{i=1}^{N} \frac{1}{2} H_i^{(t-1)} \left[h(x_i) - \left(-\frac{g_i^{(t-1)}}{H_i^{(t-1)}} \right) \right]^2 + \Omega(h(x_i))$$

The expression comes from a second-order approximation of the loss function by a Taylor expansion (as used in Newton-Raphson type optimization methods). The main details of the step-by-step development are the following:

1. **Total loss function:**

   In boosting, the objective in each iteration (t) is to minimize the total loss of the model, including a regularization term:

   $$\mathcal{L}^{(t)} = \sum_{i=1}^{N} L(y_i, F_{t-1}(x_i) + f_m(x_i)) + \Omega(f_m)$$

   where

   - L is the loss function,
   - $F_{t-1}(x_i)$ is the cumulative prediction of the model up to iteration $t-1$,
   - $f_t(x_i)$ is the new model (tree) that is being adjusted,
   - $\Omega(f_t)$ is a regularization term that penalizes the complexity of the model.

   The regularization term penalizes the complexity of the new tree added, helping to avoid overfitting. For XGBoost, the penalty is defined as:

$$\Omega(f_t) = \gamma J + \lambda \|w\|^2 = \gamma J + \lambda \sum_{j=1}^{J} w_j^2$$

where:

- J is the number of leaves on the tree.
- w_j is the value in the leaf j.
- γ and λ are hyperparameters that control the complexity penalty of the tree.

2. **Approximation by second-order Taylor expansion:**

   To make the optimization more efficient, the loss function is approximated using a Taylor expansion around $F_{t-1}(x_i)$:

$$L(y_i, F_{t-1}(x_i) + f_t(x_i)) \approx L(y_i, F_{t-1}(x_i)) + g_i^{(t-1)} f_t(x_i) + \frac{1}{2} H_i^{(t-1)} f_t(x_i)^2$$

   where:

$$g_i^{(t-1)} = \left. \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x_i)=F_{t-1}(x_i)}, \quad H_i^{(t-1)} = \left. \frac{\partial^2 L(y_i, F(x_i))}{\partial F(x_i)^2} \right|_{F(x_i)=F_{t-1}(x_i)}$$

3. **Simplified objective function:**

   Eliminating constant terms (which do not depend on f_t), the function to be minimized reduces to:

$$\sum_{i=1}^{N} \left(g_i^{(t-1)} f_t(x_i) + \frac{1}{2} H_i^{(t-1)} f_t(x_i)^2 \right)$$

4. **Reformulation as a weighted quadratic regression:**

   Using the square completion technique we can rewrite the problem so that it can be reinterpreted as a weighted quadratic regression. Thus, the optimization is rewritten as:

$$h_t = \arg\min_{h \in \Phi} \sum_{i=1}^{N} \frac{1}{2} H_i^{(t-1)} \left[h(x_i) - \left(-\frac{g_i^{(t-1)}}{H_i^{(t-1)}} \right) \right]^2 + \Omega(h(x_i))$$

This formulation allows optimizing the values assigned in each leaf of the tree in an efficient way, using both the first derivative (gradient) and the second derivative (Hessian) to obtain a more accurate update of the model at each iteration [14].