POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Reinforcement Learning-based Trading Model for US Sectors

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Author: **Giulia Mulattieri**

Student ID: 962621
Advisor: Prof. Marcello Restelli
Co-advisors: Antonio Riva, Luca Sabbioni
Academic Year: 2021-2022

Abstract

In this thesis we present an examination and implementation of a Reinforcement Learning algorithm to develop a trading model for the US market sector indices. These indices are constructed according to the Global Industry Classification Standard (GICS). We incorporate both fundamental and technical financial indicators as features in order to improve forecasting accuracy, instead of relying only on historical price series. This enables us to handle the dynamic and complex nature of financial markets more efficiently.
Our final objective is to employ the signals derived from the trading model to construct a *market-neutral portfolio*, which is a portfolio with a null net exposure with respect to the market.
We perform an extensive analysis of selected financial indicators to evaluate their predictive capabilities. We therefore identify the most informative features that can be used to train our RL algorithm. We choose Fitted Q-Iteration algorithm to train a weekly trading model on historical daily data. The performance evaluation of the model is conducted on out-of-sample data, taking transaction costs into account. We were able to develop a trading model that produced positive results for most of the sector indexes during the testing period. However, the overall trading strategy displays high volatility. Despite these instability issues, a proper implementation of volatility control techniques could potentially make this approach able to produce an effective and stable trading strategy for US sectors.

Keywords: Reinforcement Learning, Fitted Q Iteration, Financial Time Series, Investment Strategy, Market Neutral Portfolio.

Abstract in lingua italiana

In questa tesi presentiamo lo studio e l'implementazione di un algoritmo di Reinforcement Learning per sviluppare un modello di trading per gli indici dei settori del mercato americano. Questi indici sono costruiti secondo lo Standard di Classificazione Globale dell'Industria (GICS). Abbiamo considerato sia indicatori finanziari fondamentali che tecnici come variabili al fine di migliorare l'accuratezza delle previsioni, invece di affidarci solo alle serie storiche dei prezzi. Ciò ci consente di gestire in modo più efficiente la natura dinamica e complessa dei mercati finanziari. Il nostro obiettivo finale è quello di impiegare i segnali derivati dal modello di trading per costruire un portafoglio neutrale al mercato, ossia un portafoglio che ha un'esposizione netta nulla rispetto al mercato. Abbiamo effettuato un'analisi approfondita degli indicatori finanziari per valutare le loro capacità predittive. Abbiamo quindi identificato le variabili più informative da utilizzare nel nostro algoritmo di RL. Abbiamo scelto l'algoritmo Fitted Q-Iteration per costruire un modello di trading settimanale sui dati storici giornalieri. La valutazione delle performance del modello è stata condotta su dati out-of-sample, tenendo conto dei costi di transazione. Abbiamo sviluppato un modello di trading che ha prodotto risultati positivi per la maggior parte degli indici settoriali durante il periodo di test. Tuttavia, la strategia di trading complessiva presenta un'elevata volatilità. Nonostante questi problemi di instabilità, riteniamo che, con una corretta implementazione di tecniche di controllo della volatilità, questo approccio possa generare delle efficaci strategie di trading per gli indici settoriali americani.

Parole chiave: Apprendimento tramite Rinforzo, Fitted Q Iteration, Serie Storiche Finanziarie, Strategia di investimento, Portafoglio Neutrale al Mercato.

Contents

1 | Introduction

In the last decades investors have faced numerous challenges when making investment decisions. The vast number of investment options, constantly evolving market conditions, and the huge amount of available data make it increasingly difficult to perform effective investment strategies.

To overcome these challenges, investors must conduct comprehensive research and analysis of the market. In particular, opportunities and risks in investing in a specific asset can be identified by analyzing historical performance, market trends, and financial indicators. However, investing solely in a single asset can be risky, as it exposes investors to significant volatility and uncertainty. Therefore, it is crucial to use single asset analysis and market signals to construct a balanced and diversified portfolio that maximizes returns and minimizes risks.

In recent years, there has been growing interest in applying artificial intelligence (AI) techniques to assist investors in managing financial markets and making well-informed investment decisions. As highlighted in [10], AI has the potential to revolutionize trading and portfolio management by enabling investors to analyze huge amounts of data and make effective investment decisions. The article features an interview with Heloise Greeff, a mechatronics engineer at Harvard who outperformed the S&P index for the past five years. She asserts that AI improves performance while reducing the computational cost of decision-making. Therefore, by properly analyzing market trends and historical data, AI algorithms can assist investors in analyzing and selecting assets. Thus the integration of AI represents a promising opportunity to achieve optimal investment objectives.

In this thesis, our focus is on applying AI techniques to find the best investment strategy. However, before delving into the details of these techniques, it is essential to introduce the concepts of *asset* and *portfolio*. A *portfolio* is a collection of financial assets, which are instruments that represent an ownership claim on the underlying asset or a contractual right to receive future cash flows. Financial assets can take various forms, such as stocks, bonds, options, futures, and currencies. They are typically traded in financial markets, such as stock exchanges, bond markets, or commodity markets, and their values are determined by supply and demand. The composition of a portfolio can vary depending on

the investor's goals and risk tolerance. It can be structured in different ways, such as by asset class, geographic region, industry sector, or other criteria. Asset class refers to grouping assets together based on their underlying characteristics. Common asset classes include stocks, bonds, and commodities. Geographic Region approach involves selecting assets based on the country or region they belong to. For example, a portfolio might be structured to focus on US-based companies. Industry Sector method, instead, involves grouping assets by industry, such as financials, energy or technology. There are many other criteria to consider when constructing a portfolio, which can include a variety of different aspects, such as market capitalization or ESG factors [12].
However, the key principle to construct a portfolio is diversification, which means investing in different assets to manage risk by reducing exposure to any one particular security or market. Therefore it is essential to analyze and select the right assets to properly construct a portfolio able to achieve an optimal balance between performance and risk. This can be achieved through traditional approaches or AI approaches.
Traditional approaches, e.g. Markovitz model [34], typically rely on static allocation strategies based on historical time series and mathematical optimization techniques to find the best investment strategies. However, an investment strategy should be a dynamic process, as market conditions and asset performance change over time, and these approaches cannot always handle the dynamic nature of financial markets. Moreover, they usually make some strict and unrealistic assumptions about the distribution and stationarity of the data, which may be inadequate to describe the complexity and uncertainty of the financial markets.
AI approaches can overcome the limitations of traditional methods; in particular, using Reinforcement Learning, it is possible to construct an agent that can make effective investment decisions in a dynamic financial environment. The agent is trained on a series of historical data and can perform real-time optimization of its investment strategy based on available information, such as market conditions. One of the major challenges for this kind of approach is the management of non-stationarity of the data. In fact, financial markets are known for their non-stationary nature, as they often present varying behavior over time that can cause fluctuations and shifts in the underlying trends and patterns. This can pose a challenge for AI models that are trained on historical data. In this thesis, we aim to apply effective techniques for handling non-stationarity in financial data. By doing so, we aim to improve the predictive power of the features used in our models, and consequently, build a more reliable and robust model for financial analysis and decision-making.

1.1. Research goal

Most of the portfolio optimization procedures put the focus on modeling the interaction between the assets, consequently finding the allocation of all the assets in a single step and directly constructing an optimal portfolio. Instead, in this work, we will try to leverage the more detailed information that the single asset analysis can provide. We will propose an approach that exploits the representative power of reinforcement learning techniques to characterize the optimal investment strategy when dealing with a single asset. In particular, we want to build a *trading model* for the US Sectors. In a financial context, a *trading model* is an investment strategy used to generate buy and sell signals by analyzing various market indicators, such as moving averages, relative strength, or other technical indicators. The goal is to determine when to buy or sell a particular security or asset class in order to maximize returns. Therefore we aim to build a *trading model* to properly identify buy and sell signals for US Sectors. The mission of this project, that goes far beyond the contributions of this thesis, is to construct a portfolio accordingly to these signals.

Therefore our methodology relies on a two-step procedure: we first want to determine an optimal trading strategy for each sector individually by means of reinforcement learning algorithms, and secondly, optimization procedures will be performed over the obtained strategies to construct a market-neutral portfolio.

A portfolio is considered *market-neutral* if the size of the long positions equals the size of the short positions, resulting in a null net exposure against the market.

The goal is to make such portfolio, obtained from the strategies of our *trading model*, able to achieve better performance than the market weighting approach based on capitalization and an equal-weighted portfolio. The first consists of weighting stocks by their market capitalization, also known as *market cap*, which represents the total value of a company's outstanding shares of stock. It is calculated by multiplying the number of outstanding shares by the current market price per share. While market capitalization weighting can be a useful tool for constructing an index or portfolio, it can also lead to a concentration of investments in a small number of large-cap companies or sectors, potentially increasing overall risk and reducing diversification. The second one consists of weighting stocks equally, leading to a more balanced and diversified portfolio. On the other hand, this approach may also result in a lower concentration of investments in larger and more established companies that may offer a good investment opportunity.

Therefore we aim to develop an effective *trading model* through reinforcement learning techniques, which will enable us to develop an alternative weighting methodology for constructing portfolios. This methodology can potentially mitigate risk concentration

and increase overall returns.

In order to make this approach practical for real-world use, we make no simplifying assumptions. The main challenges we face are the limited available data (daily data from 2008 to 2022) and their non-stationarity.

First, we consider the main financial indicators that drive the market. We consider both technical and fundamental indicators, and through feature selection algorithms, we find the most relevant ones, i.e. those that have the greatest impact on sectors performance. Then we try to handle the non-stationarity in order to increase the predictive power without losing information.
As explained above, before focusing on identifying the optimal weights to construct a portfolio, we decide to identify individual signals for buying and selling for each asset. Therefore, through a Reinforcement Learning model, we train an agent on the selected financial variables to find the best investment strategy for every sector. By using these contextual variables, in fact, the algorithm makes more effective decisions compared to ones based on historical price series alone.

As the goal of this thesis is to be employed in a real-world portfolio management, our approach also takes into account the requirement of being easily explainable and justifiable to clients and investors. Thus, we seek a trade-off between the efficiency and transparency of the models. As a result, we decided to employ the Fitted Q-Iteration (FQI) algorithm [20] for our reinforcement learning approach. This algorithm offers the advantage of not relying on black-box methods, such as neural networks, and allows for more transparent monitoring of the decision-making process (Section 4.4). Furthermore, FQI is a highly robust model and, by discretizing actions, it mitigates potential high-dimensionality issues. To train our model, we used daily data from 2008 to 2019. Finally, we evaluated the model's performance through a backtest between 2020 and 2022. We achieve positive performance for most of the sectors for some iteration of FQI. However, some of the obtained strategies exhibit excessive volatility. One way to overcome these instability issues is to implement effective volatility control techniques which could make our trading strategy a promising approach for producing positive and stable results.

1.2. Outline of the Thesis

The thesis is organized as follows:

- in Chapter 2 we cover the theoretical background of Reinforcement Learning (RL). First we introduce Markov Decision Process, which are the mathematical framework

on which RL is based. Following this, we present the main RL algorithms, including the FQI algorithm, which will be utilized in our trading model.

- in Chapter 3 we provide an overview of the various approaches used to implement trading strategies and optimize portfolios. We first present model based and machine learning based approaches. We analyze their limitations, which lead to the advent of Reinforcement Learning based approaches.
- in Chapter 4 we cover the introduction of key financial concepts, followed by the presentation of the problem formulation of the single asset trading model.
- in Chapter 5 we first present a qualitative analysis of financial indicators, both technical and fundamental. Then we select the most relevant ones through feature selection algorithms.
- in Chapter 6 we show the achieved experimental results employing FQI algorithm with XGBoost and ExtraTrees as regressor.
- in Chapter 7 we illustrate conclusions drawn from experimental results and possible future developments.

2 | Reinforcement Learning

Machine learning is a subfield of Artificial Intelligence that involves the development of algorithms and models that can analyze data and/or make predictions. There are three different types of machine learning: supervised learning, unsupervised learning, and reinforcement learning.
Supervised learning [15] aims to train a model to make accurate predictions by using labeled datasets. Specifically, the model is trained on a given dataset where the correct output is provided for each sample. Once the model is trained, it is used to make predictions on new data.
In unsupervised learning [3], the model is not provided with training input-output examples. Instead, it must discover the structure and pattern of the data through techniques such as clustering or dimensionality reduction.

Reinforcement learning [50] develops and trains an *agent* to interact with a given system, denoted as *environment*, in order to maximize a reward signal. In particular, the agent improves its behavior through interaction with the environment and receiving feedback in the form of a scalar rewards for its actions. The agent-environment interaction is schematized in Figure 2.1, where the state $S_t \in \mathcal{S}$ represent the set of information that an agent has about the environment at a given time, $A_t \in \mathcal{A}$ represents what an agent can do in each state, r_t is the reward.

Reinforcement learning has been applied in a variety of domains, such as game playing [26], robotics [18], and finance, and is particularly well-suited for problems where there is a dynamic framework and the optimal solution must be learned through interaction with the environment.

2.1. Markov Decision Processes

The reinforcement learning theory is built on the framework of Markov decision processes [39], which are stochastic mathematical systems able to model the interaction between an

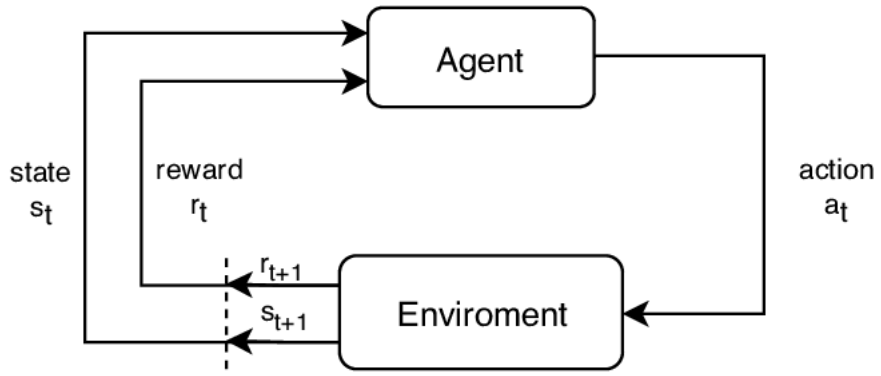

Figure 2.1: Agent-Environment Interaction Schematization

agent and an environment:

Definition 2.1.1 (Markov Decision Process)**.** *A Markov Decision Process (MDP) is a tuple* $< \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu >$*, where:*

- $\mathcal{S}$ *is the (finite) set of all possible states called state space;*
- $\mathcal{A}$ *is the (finite) set of all possible actions called action space;*
- P *is the stationary transition probability matrix defining P(s'/s,a);*
- R *is the reward function* $R(s, a) = \mathbb{E}(r|s, a)$*;*
- γ *is the discount factor such that* $\gamma \in [0, 1]$*;*
- μ *is the probability distribution over all states modeling the probability of a state to be the initial one*

In a MDP, the probability of transitioning from one state to another depends only on the current state and not on any previous states. This property is known as the Markov property and can be expressed as:

$$P(S_{t+1} = s'|A_t, S_t, S_{t-1}, A_{t-1}, .., S_1, A_1, S_0, A_0) = P(S_{t+1} = s'|S_t, A_t), \tag{2.1}$$

where $P(S_{t+1}|S_t, A_t)$ is called *Transition Probability*.

By using a MDP to model the environment, the agent can use the transition probabilities to estimate the expected rewards of different actions in each state and find the optimal policy, which means the policy that maximizes its cumulative reward over time.

2.2. Policy

A policy π is a function that maps states to a probability distribution over the action space, and it can be expressed as:

$$\pi(a|s) = P(A_t = a|S_t = s). \tag{2.2}$$

It represents the behavior of the agent, in particular how the agent selects actions based on the current state.
In a MDP it is possible to exclusively consider stationary and Markovian policies (depending only on the current state and action) since it is demonstrated that it always exists an optimal policy with these characteristics. Moreover, a policy can be stochastic or deterministic.
The *agent*, at each state, has to make a decision following a certain policy and then receive feedback, giving rise to a sequence of states S, actions A, and rewards R known as a trajectory:

$$s_0, a_0, r_1, a_1, r_2, ... \tag{2.3}$$

2.3. Return

The goal of the agent is to maximize the cumulative reward, that can be expressed as the sum of all the rewards received over time. In RL there exist different reward functions used to measure the cumulative reward:

- Total Reward V = $\sum_{i=0}^{\infty} r_{i+1}$
- Average Reward V = $\lim_{n\to\infty} \frac{r_1+...+r_n}{n}$
- Discounted Reward V = $\sum_{i=0}^{\infty} \gamma^i r_{i+1}$, where $\gamma \in$ [0,1] is the discount factor. In particular, γ represents the importance of future rewards. A value of γ closer to 1 implies that the agent places more emphasis on future rewards, while a lower value of γ indicates a greater focus on immediate rewards. Therefore, the choice of γ is critical in balancing the trade-off between short-term and long-term gains in a reinforcement learning system.

The most commonly used reward in RL is the Discounted Reward since it avoids divergence issues. From the definition of Discounted Reward, it is possible to introduce a key

element in RL, the *return*:

$$v_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.4}$$

2.4. Value Functions

The value function V of a state s represents the goodness of that state following a specific policy and informs the agent of how much return to expect if it takes an action in that state. It's computed as a prediction of expected future rewards:

$$V_\pi(s) = \mathbb{E}_\pi[v_t | S_t = s]. \tag{2.5}$$

Where s is the current state at time t, π is the policy followed by the agent, r is the reward and

The action-value function Q_π describes the expected future reward for taking a particular action a in a given state s following a certain policy π. It can be expressed as:

$$Q_\pi(s, a) = \mathbb{E}_\pi[v_t | S_t = s, A_t = a]. \tag{2.6}$$

One of the key tools in reinforcement learning are the Bellman equations [50], used to describe the relationship between the value of a state or action and the expected future return by following a specific policy. They are named after Richard Bellman, who introduced them in the 1950s.
The first one is a mathematical equation that describes the relationship between the value V_π of a state s (or a state-action pair) and the expected value of the next state. It helps the agent to decide which action to take in order to maximize the long-term return. Therefore, starting from the definition of the value function, it is possible to establish a recursive relationship between the expected returns of consecutive states, leading to the Bellman Equation:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s)(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_\pi(s')). \end{aligned} \tag{2.7}$$

2.5. Optimal Value Functions

The optimal state-value function $V^\star(s)$ is the maximum state-value function over all policies:

$$V^\star(s) = \max_\pi V_\pi(s). \tag{2.8}$$

The optimal action-value function $Q_\star(s, a)$ is the maximum action-value function over all policies:

$$Q^\star(s, a) = \max_\pi Q_\pi(s, a). \tag{2.9}$$

Value functions also define a partial ordering over policies:

$$V_\pi(s) \geq V_{\pi'}(s)\ \forall s \in \mathcal{S} \implies \pi \geq \pi', \tag{2.10}$$

which means that it is possible to define the optimal policy ($\pi^\star$) as the policy that is always better or equal than any other policy. The optimal policy is not unique in an MDP but all optimal policies lead to the same (optimal) value functions $V^\star(s), Q^\star(s, a)$ $\forall s, a$.

The relationship between the optimal value function and the optimal policy is characterized by the *Bellman Optimality Equation*. It states that the optimal value function can be expressed as the maximum expected return achievable by following any policy:

$$\begin{aligned} V^\star(s) &= \max_{a \in \mathcal{A}} Q_{\pi^\star}(s, a) \\ &= \max_{a \in \mathcal{A}} (R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^\star(s')] \end{aligned} \tag{2.11}$$

This equation essentially states that the optimal value function satisfies a recursive relationship that takes into account the expected immediate reward and the expected value of the next state under the optimal policy.

The Bellman equations provide a fundamental framework for evaluating and optimizing decision-making processes. In particular, these concepts form the basis for reinforcement learning algorithms, which enable agents to efficiently make optimal decisions by interacting with the environment.

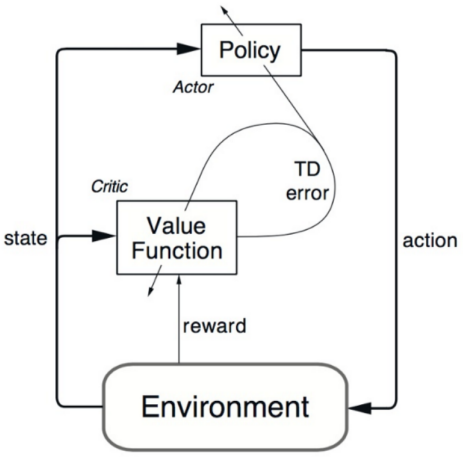

Figure 2.2: Actor Critic Schematization

2.6. Algorithms

Reinforcement learning algorithms can be divided into two main categories: value-based and policy-based. The former tries to learn the optimal policy by iterating on the space of value function thanks to the Bellman Optimality equation, while the latter learns the optimal policy by iterating over the space of the policies. In the first case, the algorithm estimates the state value (usually the value function or action value function), while in the second case, it directly obtains the optimal policy without estimating the value of each state by using a parameterized function to represent the policy, which is optimized through gradient ascent to maximize the expected return.

There are also actor-critic algorithms that are a compromise between policy-based and value-based. They simultaneously perform two learning tasks: one for optimizing the policy using gradient descent, and another for estimating the advantage function A (Equation 2.12), which can be expressed as:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s). \tag{2.12}$$

In particular, the actor takes as input the state and outputs the best action using Policy Gradient (Section 2.8) with estimations from the critic. It essentially controls how the agent behaves by learning the optimal policy (policy-based). The critic, on the other hand, evaluates the current policy by computing the value function (value-based) and the result is used in the policy training. Therefore these algorithms are able to learn both value function and policy.

In the following sections, we introduce Q-learning and Policy Gradient, which are two fundamental reinforcement learning algorithms that have a significant impact on the development of many other RL algorithms. Q-Learning and Policy Gradient, in fact, have been instrumental in advancing the field of RL by providing a foundation for exploring more advanced and complex RL techniques, such as Actor-Critic and Deep Reinforcement Learning.
In particular, Q-learning is a value-based method that involves learning an action-value function (Q-function) to estimate the expected cumulative return of taking a particular action in a given state, while Policy Gradient is a policy-based method that involves learning a policy function directly.

2.7. Q-Learning

The Q-learning approach [59] involves an agent learning the Q-value function by updating a table of action values called the *Q-table* (Figure 2.3). The Q-table stores the expected return for each action in each state, and the agent uses this information to determine the optimal action to take in any given state. The Q-function is learned through an iterative process of updating the Q-values based on the Bellman Equation. Once the Q-function is learned, the agent can use it to select the optimal action at each state by choosing the action with the highest Q-value. This allows the agent to learn a policy that maximizes the expected cumulative reward over time.
The Q-Learning pseudocode is presented in Algorithm 2.1, where α is the learning rate, which is the parameter used to control the degree to which the Q-values are updated in each iteration of the algorithm. Specifically, it determines the weight given to the new information obtained from the observed reward and the estimated maximum future reward. A high learning rate means that the Q-values are updated more strongly in response to new information, while a low learning rate means that updates are more gradual and stable.

As previously mentioned, Q-learning employs a tabular representation of the Q function, which makes it well-suited for tabular environments. However, to extend this approach to more complex environments, an approximation method is necessary. Two popular variations of the Q-learning algorithm, presented in the next sections, are Fitted Q-Iteration (FQI) and Deep Q-Network (DQN). FQI is an extension of Q-learning that utilizes batch learning to estimate the Q-function. With batch mode learning, the agent does not directly interact with the environment, instead, it is provided with a set of

Algorithm 2.1 Q-Learning

1: Initialize the Q-table arbitrarily with Q-values Q(s,a) for each state-action pairs
2: Iterations:
Repeat until stopping conditions are reached:
- Choose an action (a) in the current state s based on the current Q-value estimates Q(s,·)
- Take the action a and observe the next state s' and the reward r
- Update

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{2.13}$$

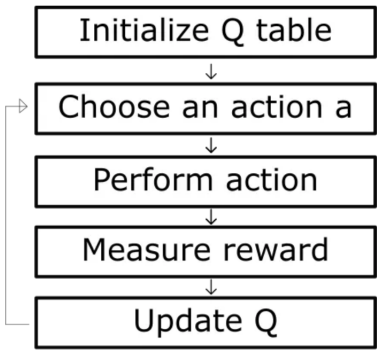

Figure 2.3: Q learning Schematization

tuples. The agent's task is to evaluate these tuples and deduce a control policy that closely approximates the optimal policy.
DQN is a variant of Q-learning that utilizes deep neural networks to approximate the Q-function.

2.7.1. Fitted Q-Iteration

Fitted Q-Iteration (FQI, [20]) is a value-based algorithm that aims to approximate the optimal action-value function for each action in a given state. The core idea of FQI is to use an approximate model of the system dynamics to estimate the action value function. The model is iteratively trained on a dataset in the form (s, a, s', r) with the aim of obtaining the optimal policy. Unlike Q-learning, which uses a lookup table to store Q-values, FQI uses a regression model, such as Extra Trees [22] or XGBoost [13], to estimate the Q-values for each state-action pair. This iterative process updates the model based on a batch of experiences collected from the environment, allowing it to learn from the past and improve its predictions of future rewards in order to obtain a new policy. The final result is an approximation of the optimal policy that maximizes the long-term reward.

Algorithm 2.2 FQI

1: **Input**: a set of four-tuples $\mathcal{F}$ and a regression algorithm
2: **Initialization**: set N equal to 0. Let $\hat{Q}_N$ be a function equal to 0 everywhere on S×A
3: **Iterations**:
Repeat until stopping conditions are reached:
- N ← N+1
- Build the training set $\mathcal{TS} = (i^l, o^l), l = 1, ..., \#\mathcal{F}$ based on the function $\hat{Q}_{N-1}$ and on the full set of four-tuples $\mathcal{F}$:

$$\begin{aligned} i^l &= (s_t^l, a_t^l), \\ o^l &= r_t^l + \gamma \max_{a \in \mathbb{A}} \hat{Q}_{N-1}(s_{t+1}^l, a) \end{aligned} \tag{2.14}$$

- Use the regression algorithm to induce from $\mathcal{TS}$ the function $\hat{Q}_N(s, a)$

One key advantage of using a machine learning model like Extra Trees or XGBoost is that it can capture complex, non-linear relationships between states and actions, which can be difficult to represent using a simple table. Another advantage is that the model can generalize to unseen states, allowing it to make accurate predictions even in situations where it has not seen a particular state before.

2.7.2. Deep Q-Network

Deep Q-Network (DQN, [35]) algorithm is a value-based algorithm that adopts a deep neural network to predict the expected cumulative reward and, therefore, an approximation of the Q-function for each action in a given state. During training, the agent interacts with the environment and stores transitions (state, action, reward, next state) in a replay memory. At each step, the agent selects an action with the highest predicted Q-value according to the current neural network, with some probability of selecting a random action to encourage exploration. The agent then executes the selected action and observes the resulting reward and next state. To update the neural network, DQN employs the Bellman equation to compute the target Q-value for each transition, which is a combination of the immediate reward and the estimated maximum Q-value for the next state. The target Q-values are then used to train the neural network to minimize the mean squared error between the predicted and target Q-values.

Algorithm 2.3 DQN

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for *episode* $= 1, ..., M$ **do**
  Initialize sequence $s_1 = x_1$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, ..., T$ **do**
    With probability ϵ select a random action a_t otherwise select $a_t = \max_a Q^{\star}(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $st + 1 = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

2.8. Policy Gradient

Policy gradient [47, 51] is a popular approach in reinforcement learning for finding the optimal policy in a Markov decision process (MDP). It involves directly optimizing the parameters of a policy function by using gradient-based optimization methods.
We define a set of parameters θ to parametrize this policy π_θ. If we represent the total reward for a given trajectory (Equation 2.3) τ as $r(\tau)$, the objective is to find the parameters that maximize the expected cumulative reward over a set of trajectories following a parametrized policy:

$$J(\pi_\theta) = \mathbb{E}_\pi[r(\tau)]. \tag{2.15}$$

This is typically done by using stochastic gradient ascent (or descent) to iteratively update the parameters of the policy function, based on estimates of the gradient of the objective function. In particular, in gradient ascent, the parameters are updated using the following rule:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi_{\theta_t}). \tag{2.16}$$

This optimization procedure can be performed by using the policy gradient theorem. It states that the gradient of the expected cumulative reward with respect to the policy parameters can be expressed as an expectation over the state-action distribution under

the policy:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim p_\theta, a \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s_t, a_t) \right], \tag{2.17}$$

where θ are the parameters of the policy π_θ, $J(\pi_\theta)$ is the expected reward under the policy, p_θ is the distribution over trajectories induced by the policy,
$\tau = \{(s_0, a_0, r_0), (s_1, a_1, r_1), ..., (s_T, a_T, r_T)\}$ is a trajectory, $Q^\pi(s_t, a_t)$ is the expected return starting from state s_t and taking action a_t under the policy π_θ), and ∇_θ denotes the gradient with respect to the policy parameters θ.
This result allows for an unbiased estimate of the gradient, which can be used to update the policy parameters. PPO (Proximal Policy Optimization) and TRPO (Trust Region Policy Optimization) are policy gradient-based algorithms that include constraints to improve their stability and convergence. In particular, both algorithms use a constraint on the magnitude of policy parameter updates during neural network weight updates. TRPO uses a constraint on the Kullback-Leibler (KL) divergence between the old and new policies, while PPO uses a constraint on the ratio between the consecutive policies. These constraints are designed to avoid overly drastic policy parameter updates, which can cause instability or a deterioration in performance. Additionally, these constraints help ensure that the updated policy remains close to the old policy, which can be important for preventing the loss of previously learned useful information.

2.8.1. Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO, [43]) algorithm is a policy-based algorithm that aims to find the optimal policy. In particular, it employs a constraint on the maximum change in policy distribution (KL divergence) to ensure that the new policy does not deviate too far from the previous policy. This constraint helps to ensure that the new policy is still close enough to the old one allowing effective policy updates in a robust way.

This means that we have a trust region constraint of the following kind:

$$\begin{cases} \min\limits_{\theta} L_{\theta_{old}}(\theta) \\ D_{KL}^{max}(\theta_{old}, \theta) \leq \delta \end{cases} \tag{2.18}$$

where $L_{\theta_{old}}(\theta)$ is a local approximation of the expected return obtained with a new policy θ under the experience collected by the old policy θ_{old}. This problem is practically unsolvable due to a large number of constraints. Instead, we can solve an approximation of this problem which considers the average KL divergence as a constraint.

Algorithm 2.4 TRPO

Input: initial policy parameters θ_0
Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking test n_{cg}
for $k = 0, 1, 2, ...$ **do**
  Collect set of trajectories D_k on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Form sample estimates
    - policy gradient $\hat{g}_k$ using advantage estimates
    - KL-divergence Hessian-vector product function f(v) $= \hat{H}_k v$
  Use the conjugate gradient algorithm with n_{cg} iterations to obtain $x_k \approx \hat{H}_k^{-1}\hat{g}_k$ where $\hat{H}_k$ is the Hessian of the sample average KL-divergence
  Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$
  Perform backtracking line search with an exponential decay to obtain the final update
$$\theta_{k+1} = \theta_k + \alpha^j \Delta_k$$
end for

The algorithm adopts a surrogate objective function that approximates the performance improvement of the new policy:

$$L(\theta) = \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \tag{2.19}$$

where $\hat{A}_t$ is an estimate of the advantage function A_t (Equation 2.12).
Then a constrained optimization on this function is performed using a conjugate gradient method. This optimization step helps to ensure that the new policy update is guaranteed to improve the performance of the agent.

2.8.2. Proximal Policy Optimization

Proximal Policy Optimization (PPO, [45]) algorithm is a policy-based algorithm that provides an improvement on TRPO. It works by iteratively updating the policy of the agent based on a surrogate objective function, which provides a lower bound on the performance improvement of the new policy update. A commonly used surrogate objective function is the clipped one, which helps to prevent the policy from changing too much between updates and provides a more stable optimization process.

In particular, TRPO maximizes the objective in Equation 2.19, which can lead to large

Algorithm 2.5 PPO Actor-Critic Style

for *iteration* $= 1, 2, ...$ **do**
  for *actor* $= 1, 2, ..., N$ **do**
    Run the policy π_{old} in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size M $\leq$ NT
  $\theta_{old} \leftarrow \theta$
end for

Algorithm 2.6 PPO with Clipped Surrogate Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, ...$ **do**
  Collect set of partial trajectories D_k on policy $\pi_k = \pi(\theta_k)$ in the environment
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update by maximizing

$$\theta_{k+1} = \arg\max_{\theta} L_{\theta_k}^{CLIP}(\theta)$$

  via stochastic gradient ascent with Adam, where

$$L_{\theta_k}^{CLIP}(\theta) = \hat{\mathbb{E}}_{\tau \sim \pi_k}[\sum_{t=0}^{T}[\min(r_t(\theta)\hat{A}_t^{\pi_k}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{\pi_k})]]$$

end for

policy updates. PPO instead wants to maximize:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon))], \tag{2.20}$$

where $r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. The *clip* function defines an interval for r_t, which is clipped into a range [1-ϵ, 1+ϵ], where ϵ is a hyperparameter. This is done in order to penalize changes to the policy that make r_t far from 1.

It employs the advantage function $\hat{A}$, estimated through generalized advantage estimation (GAE, [44]), instead of the expected return in order to reduce the variance of the estimations.

3 | Related Works

Efficiently constructing a portfolio requires properly analyzing the assets' historical performance and correctly identifying market signals. This is a crucial topic in finance and, in order to obtain the best investment strategy, several issues need to be addressed including handling the non-stationarity of the data, balancing risk and return, and finally dealing with the high dimensionality of the portfolio space. Therefore several approaches can be found in recent literature.

3.1. Model based approaches

With *model-based approaches*, we refer to a broad family of mathematical techniques that are used to determine the optimal allocation of assets, individually or in a portfolio, based on a set of constraints and assumptions. These methods rely on historical data, and are designed to optimize a specific criterion such as the Sharpe Ratio [46].
One popular approach is the mean-variance (MV) optimization technique, also known as Modern Portfolio Theory (MPT), which is a method for constructing a portfolio that maximizes the expected return for a given level of risk. This approach was first introduced by Harry Markowitz in [34]. Initially, the analysis focuses on individual stocks or financial assets, evaluating the historical returns, volatility, and correlation. Using this information an efficient frontier of portfolios is constructed and then the portfolio that optimizes the trade-off between expected return and risk is selected. This method is based on the assumption that investors are risk-averse: this means that every investor wants to maximize the expected return for a given level of risk.
Other two traditional approaches are Risk Parity [54] and Kelly Criterion [41]. The former estimates the expected risk of each asset and then distributes it evenly to construct a portfolio. This means that higher-risk assets receive smaller allocations than lower-risk assets, with the goal of achieving a more balanced risk profile across the portfolio. This approach is based on the idea that the best performance is obtained when the risk of a portfolio is allocated in proportion to the volatility of each asset. Kelly Criterion, instead, aims to maximize the long-term growth while minimizing the overall risk. In particular,

it maximizes the expected logarithm of the portfolio value, representing wealth, which means maximizing the expected value of the portfolio growth rate. It can be proven that, under certain conditions, there is an equivalence of the Markovitz model with Kelly Criterion or Risk-Parity [5]. This is an important result, leading to a universal solution to the problem of portfolio optimization.
The main limitation of these approaches is that they are built on assumptions that may not accurately portray realistic situations: Markovitz's model assumes that all investors think rationally and avoid risks, which is not always true as some investors are high risk-takers. Additionally, it assumes that asset returns are random variables with known expected returns and variances. But in general financial time series are non-stationary and it may be difficult to accurately estimate the expected returns and variances of the assets.
Therefore, since these methods make some unrealistic assumptions which make them not suitable for real-world applications, new approaches for portfolio optimization have been developed, such as Machine Learning and Reinforcement Learning.

3.2. Machine Learning based approaches

By relaxing certain model-related assumptions, machine learning (ML) can overcome the limitations of purely model-based approaches. However, this family of techniques depends more heavily on data quality, which may lead to some stability issues as a trade-off. One potential application of ML consists in better identifying market signals and determining the key factors that influence stock performance. This information can be valuable in improving the accuracy of return predictions and overall investment strategy.

[33] employs deep learning models to predict returns for single assets. Then it constructs a portfolio employing a mean–variance (MV) approach, introduce in Section 3.1. The performance evaluation is based on historical data of 9 years from 2007 to 2015 of 100 stocks from Chinese market. Experiments show that the proposed approach outperforms traditional approaches. However, this method only relies on a window of the previous 60 daily returns to predict the next daily return which, if used alone, may not have a stable predictive power over different market conditions.

[14] combines supervised learning approaches to make returns' prediction and then MV technique, introduce in Section 3.1, to select the best portfolio. For stock price forecasting XGBoost is employed. Then the mean-variance model is employed for portfolio selection, based on the selected assets. This work randomly selects 24 stocks in the SSE 50 index as candidate assets. Additionally, 19 indicators are used as predictive features, including 15

lagged return observations and 4 technical indicators (e.g. the relative strength index). Transaction costs are considered during the performance evaluation. The obtained results demonstrate that the proposed method is superior to traditional methods (without stock prediction) and benchmarks in terms of returns and risks. The main limitations of this approach are that it considers only a few financial indicators. Moreover, this paper presents a limited backtest which does not allow us to understand how the model is actually performing.

[16] employs an artificial neural network to produce a set of continuous buy and sell signals for stock trading. Specifically, the proposed approach integrates technical analysis with machine learning techniques for the efficient generation of stock trading decisions. This approach considers three class values representing the buy, hold and sell signals. The performance evaluation is made on five years of historical stock index price values of two stock indices (BSE SENSEX and S&P 500). The results show that the model outperforms some other machine learning techniques such as Support Vector Machines (SVM), K-nearest neighbor (KNN), and Decision Tree models. However, this paper only gives as results a sell/buy signal analysis and does not provide a backtest by simulating a trading strategy employing that signals, in which also transaction costs are considered. Moreover, the proposed model is tested on limited scenarios.

Additionally, neither of the papers provides a detailed analysis of how to handle the non-stationarity of the data.
In general the main limitations of Machine Learning techniques is that they require a fixed dataset to train and they are not able to handle dynamic and uncertain environments, such as financial markets. In Reinforcement Learning, instead, an agent learns to make decisions by interacting with the environment and receiving feedback in the form of rewards: this allows the agent to adapt to changing market conditions and optimize its decisions over time, which makes it more suitable for finding the best investment strategy.

3.3. Reinforcement Learning based approaches

The advent of Reinforcement Learning has allowed us to overcome the assumptions and limitations of previous approaches by managing the dynamics and non-stationarity of financial markets more carefully, leading to better performance. RL, in fact, has been shown to be effective in dealing with dynamic and uncertain environments, making it well-suited for financial markets.
A commonly employed approach is by means of deep RL techniques, which use neural networks to approximate the agent's decision-making process. This approach can be

performed both to directly construct an optimal portfolio and develop a single asset trading strategy.

[24] presents a deep reinforcement learning approach to approximate the optimal policy and find the optimal portfolio weights. Performance is evaluated in three back-test experiments trading periods of 30 minutes in a cryptocurrency market. Despite a 0.25% commission rate in back-tests, the framework is capable of generating returns that are at least four times the initial investment within 50 days. The main limitation of this work is that it often leads to extreme weights [42] which makes the resulting optimal portfolio not diversified. This is because the reward function doesn't include any kind of penalization by portfolio volatility. Moreover, these extreme weights make this approach unstable and suitable only for buy-and-hold strategies.

[23] employs a Convolutional Neural Network in order to approximate the optimal policy. The reward function is the logarithm of the expected return of a portfolio of 100 randomly selected stocks of the CSI300 index. Positive results in terms of performance are reached, compared to some traditional strategies, on several backtests. In this case, the volatility is taken into consideration, but in the evaluation of the performances there are no transaction costs, which does not represent a realistic setting.

[48] employs a deep reinforcement learning framework called Deep-Breath. Such a methodology combines a restricted stacked autoencoder in order to conduct dimensionality reduction (only the most informative abstract features are kept) and a Convolutional Neural Network. It implements a settlement system on a Bitcoin blockchain to mitigate settlement risk. According to the results, the presented approach outperforms current expert investment strategies in terms of return on investment while minimizing market risk. The main limitation of this paper is that it is based on a blockchain simulator so there is limited knowledge on its potentiality in a real-world scenario; moreover, the authors do not provide a realistic backtest since they only evaluate their method in a very short time horizon (90 days).

[25] aims to optimize a two-stock portfolio using a Q-learning approach with an artificial neural network. As features, only historical data of the two assets are taken into consideration and transaction costs are considered during the performance evaluation. After investigating various reward functions and hyperparameters, the presented models have achieved performance similar to their corresponding benchmarks. The main limitations of this approach is that it has only 2 stocks, which may not be representative of more complex portfolio management problems which require a larger number of assets.

[1] employs deep recurrent reinforcement learning techniques to construct a real-time optimal portfolio with continuous actions and 15-minute price changes for ten selected stocks from different sectors of the S&P500 as input data. According to the experiments, the proposed approach outperforms the selected benchmark. The main limitation of this work is the instability of a continuous framework, which may lead to suboptimal solutions. Moreover only historical price series are taken into account to evaluate the next allocation, which may not be enough to describe the market conditions and make good investment decisions.

[30] experiments policy gradient, deep deterministic policy gradient, and PPO to find the best investment strategy. The authors consider 5 Chinese stocks, taking into account the volatility to penalize the agent's rewards. The experiments show that the strategy obtained by the Policy-Gradient based algorithms can outperform Uniform Constant Rebalanced Portfolios in asset allocation. However they do not consider the non-stationarity of the data, therefore the results are highly sensitive and the performance is very unstable. Furthermore, the model often leads to the purchase of only one asset at a time.

[31] presents a deep multi-agent approach which considers, as assets, the 11 sectors according to the Global Industry Classification Standard (GICS). Each agent is equipped with deep policy networks to optimize the risk-adjusted criterion. Transaction costs are considered in the performance evaluation, which consists of a 2-year backtest. As input data, only the (normalized) 1-min historical price series are considered. They obtain good results in terms of performance. However, also in this case, they do not consider technical indicators as additional features. Moreover this method is only tested with high-frequency data and it may not be well suited for different kinds of datasets, such as daily data.

[53] adopts deep reinforcement learning to generate buy and sell signals for various financial instruments including stocks, currencies, and cryptocurrencies. The authors employ the SARSA algorithm and a neural network framework based on the DQN model to find an optimal trading strategy. The proposed method is tested on real-world financial daily data and achieved superior performance compared to other state-of-the-art models in learning trading rules specific to individual assets. The proposed model also achieved almost 12.4% more profit compared to the best state-of-the-art model on the Dow Jones Index. However there is no analysis of which financial indicators should be used to better train the agent and improve the models performance.

[55] presents a Double DQN setting with Sharpe ratio as a reward function to perform daily stock trading. They consider 15-min historical data, in particular open, high, low, and close prices and the Relative Strength Index. Transaction Costs are considered in

the performance evaluation. According to the results, the proposed model is designed to assist traders in making quick and effective decisions, with the most up-to-date information from the market. Therefore this approach is shown to perform well only with high-frequency input data. Moreover, they consider, as a technical indicator, only the RSI.

An alternative approach that does not rely on neural networks is the FQI algorithm, a variation of Q-learning introduced in Section 2.7.1, which can be used used for single-asset trading with action discretization. Rather than using neural networks, FQI typically adopts regressors such as Extra Trees or XGBoost in the model. This approach enables a more transparent decision-making process and mitigates the instability issues frequently encountered in continuous framework models.

[8] and [40] develop an effective high frequency FX trading strategy by training an agent via Fitted Q-Iteration. They take into account both the non stationarity of the data and the transaction costs. However they do not incorporate any financial indicators as features.

In summary, the main limitations of current reinforcement learning approaches include:

- Limited backtesting: Some papers may not provide a realistic backtesting of their proposed methods. This leads to overfitting, unrealistic results, and a lack of generalization.
- Model complexity: Many of these works use deep learning-based models, which can be complex and difficult to interpret. This makes it challenging to understand how the model is making decisions and to identify potential errors or biases in the model. Moreover, in real-world financial framework, models have to be explained to investors so they should not be too complex or black-box.
- Real-world applicability: A large number of papers is mostly focused on simulating investment strategies considering a few assets, and it is not clear how well these methods would perform in a real-world setting where portfolios must be more diversified. Some papers also use simplified assumptions, such as the absence of transaction costs, which is not realistic.
- Non-stationarity: financial time series are non-stationary, and many papers do not provide a comprehensive solution to it.
- Feature selection: another limitation in many works consists in the adoption of a limited number of financial indicators, which may not be enough to capture all the important information about the assets and properly train the agent.

In this thesis, we propose an approach that manages daily signals to perform weekly trading without any simplified hypothesis. To achieve this goal, we incorporate effective techniques for managing data non-stationarity and selected financial indicators as input data in order to better handle the dynamics and complexity of the financial market. We adopt the FQI algorithm with action discretization as Reinforcement Learning model. Performance evaluation is made through a realistic backtest on out-of-sample data considering transaction costs.

4 | Problem Formulation

In order to effectively apply reinforcement learning algorithms to find the best investment strategy, the first step is to model the trading dynamic as a Markov decision process and define the concepts of state, action, and reward.
Our goal is to create an agent that utilizes RL techniques to perform portfolio management and/or generate trading signals. Therefore, we need to introduce fundamental concepts related to price time series and define the underlying dynamics.

4.1. Financial Preliminaries

The price of an asset is one of the fundamental measures in finance, as it represents the current value at which the asset is traded in the market. Therefore it is important to clearly specify what is meant by the price of an asset and how it is evaluated to analyze its performance.
A trading session is a period of time during which financial markets are open for buying and selling securities. Different markets around the world have their own trading sessions, which usually last for a few hours each day. For instance the American Market trading sessions, known as *New York Stock Exchange (NYSE) session*, runs from 3:30 PM to 10:00 PM in Central European time (CET), Monday through Friday.
The closing price of an asset is the price of the last transaction made on that security during a trading session. The closing price is used as a reference point to evaluate the performance of the stock during the trading session in question.
The opening price of an asset is the price of the first transaction made on that security during a trading session. Typically, the opening price is calculated using the closing price of the last trading session as a reference point.
The performance of an asset is evaluated by calculating its return [1], which can be determined in various ways depending on the time horizon under consideration. For instance,

[1] This concept of return is different from the return commonly used in RL

if we consider daily returns, they are computed as:

$$r_t = \frac{ClosePrice_t - ClosePrice_{t-1}}{ClosePrice_{t-1}} = \frac{ClosePrice_t}{ClosePrice_{t-1}} - 1 \tag{4.1}$$

Investors can buy, sell or hold financial assets, depending on their investment objectives and market conditions:

- 'Buy' means purchasing a financial asset, hoping that its value will increase in the future.
- 'Sell' involves disposing of a financial asset and selling it, either to realize a profit or to cut losses.
- 'Hold' refers to maintaining the current position with respect to the financial asset, which is typically done when investors expect that the costs of a potential transaction will be higher than future returns.

These actions represent what an investor can do with financial assets. Another crucial concept to consider is the position that an investor can have in relation to an asset. There are two main positions: 'Long' and 'Short'.
The former term refers to a position that investors can take in a financial asset in which they are hoping for an increase in the asset's value. A long position is taken by buying the asset or through an options contract that gives the investor the right to buy the asset at a specified price.
'Short' position involves selling a financial asset that the investor does not own, hoping that its price will go down in the future. If the price does fall, the investor can buy the asset back at a lower price and realize a profit. Short selling is typically used in bearish market conditions. The main difference between selling and shorting is that selling involves disposing of an asset that an investor already owns while shorting involves selling an asset that the investor does not own with the hope of buying it back at a lower price in the future.

4.2. Reward

In reinforcement learning, the reward function maps a state or a state-action pair to a scalar value that represents the 'goodness' of that state or state-action pair in terms of immediate reward. The goal of the agent is to maximize the cumulative reward it receives over time, so the reward function plays a central role in defining the agent's objective.
Our objective is to identify the optimal investment strategy for every sector (Section

5.1), with the ultimate goal of building a market-neutral portfolio. Therefore we have to properly define a reward function in order to reach this goal.

Let us consider a set Ω of financial assets. A *portfolio* is a combination of such assets, in a way that each asset $i \in \Omega$ at a specific timestep t is associated to a weight denoted as $\omega_{i,t} \in [0, 1]$ and $\sum_i \omega_{i,t} = 1$.
The value of such *portfolio* can be expressed as:

$$V_t = \sum_{i \in \Omega} \omega_{i,t-1}\left(\frac{OpenPrice_{i,t}}{ClosePrice_{i,t-1}} - 1\right) + \omega_{i,t}\left(\frac{ClosePrice_{i,t}}{OpenPrice_{i,t}} - 1\right), \tag{4.2}$$

where *OpenPrice* and *ClosePrice* are respectively the opening price and the closing price of the asset i.
Assuming that the closing price at time t-1 is equal to the opening price at time t, we obtain:

$$V_t = \sum_{i \in \Omega} \omega_{i,t}\left(\frac{ClosePrice_{i,t}}{ClosePrice_{i,t-1}} - 1\right), \tag{4.3}$$

which can be rewritten as:

$$V_t = \sum_{i \in \Omega} \omega_{i,t} r_{i,t}, \tag{4.4}$$

where $r_{i,t}$ is the return of the asset i at time t, introduced in Equation 4.1

In real markets, when an investor buys or sells an asset, a transaction cost must be paid. If we assume that the transaction costs are linear in the change of allocation, homogeneous in time and constant for all assets, the value of a portfolio including transaction costs at time t can be defined as:

$$V_t = \sum_{i \in \Omega} \omega_{i,t} r_{i,t} - fee \cdot |\omega_{i,t} - \omega_{i,t-1}|, \tag{4.5}$$

where fee is the transaction cost associated with each operation.
Given this result, we can define the reward function of the asset i as the difference between the return and the transaction cost associated with the trading operation:

$$R_{i,t} = \omega_{i,t} r_{i,t} - fee \cdot |\omega_{i,t} - \omega_{i,t-1}|. \tag{4.6}$$

In our case, we decide to work with discrete actions instead of continuous weights ω, therefore the final reward function of asset i at time t becomes:

$$R_{i,t} = a_{i,t} r_{i,t} - fee \cdot |a_{i,t} - a_{i,t-1}|, \tag{4.7}$$

where $a_{i,t} \in \mathcal{A}$ is the action performed by the agent as described in (4.8).

Our approach involves using a reward function that evaluates the performance of each asset individually, rather than treating the entire portfolio as a single entity. We make this choice because we aim to analyze each sector in a more specific and accurate way to better identify buy and sell signals, rather than directly constructing a portfolio by finding the optimal weights.
To achieve this goal, we employ the algorithm described in section 4.4, which operates at the asset level. Specifically, we construct a single asset optimal trading strategy for each sector in which the goal of the agent is to maximize the reward expressed in (4.7) over time.

4.3. Environment Formulation

In reinforcement learning, the environment is the system that the agent interacts with to learn how to perform a task and make the right decisions. At each time step, the agent receives an observation of the current state of the environment and chooses an action to take based on its current policy. The environment then transitions to a new state based on the chosen action and provides a reward signal to the agent.
In order to construct an environment suitable for a trading framework, we need to properly define the reward function (introduced in Section 4.2), the action space, and the state space.

Action Space The action consists of the allocation the agent chooses for the week, therefore the set of possible actions is defined as:

$$\mathcal{A} = \{-1, 0, 1\}, \tag{4.8}$$

which corresponds respectively to the three different possible actions described in Section 4.1: Sell, Hold, and Buy.

Feature Space The state in an MDP contains all information needed to select the best action to maximize future rewards. While the current state of an asset is trivially represented by the current price, we can include a series of financial indicators that may provide useful information to better understand the underlying dynamics of the dynamic

process. These indicators can be obtained by considering the past asset prices, or by gathering market information. The most significant financial indicators we select are:

- Fundamental indicators: These indicators are related to a single company. The corresponding sector indicator is constructed by considering the indicators of the companies in that sector.

    1. Price to Book (PB): it is a financial indicator computed by dividing the price of a company by its total book value, which is the difference between its assets and its liabilities. It indicates whether a stock is undervalued or overvalued. In particular, a low PB ratio means that the stock is undervalued and may be a good investment opportunity, while a high PB ratio indicates that the stock is overvalued and may be overpriced.

    2. Price over earnings (PE): it is a financial indicator computed by dividing the price of a company by its earnings per share (EPS) from the last 12 months. This ratio is used to determine whether a stock is undervalued or overvalued. A high PE ratio indicates that the stock is overvalued and may be overpriced, while a low PE ratio indicates that the stock is undervalued and may be a good investment opportunity.

    3. Dividend yield: it is a financial indicator computed by dividing the annual dividends per share (DPS) of a company by the share price. It is used to evaluate the income of a stock. In particular, a high or low dividend yield indicates that a stock is paying a high or low level of dividends relative to its market price.

    4. Earnings growth: it is a financial indicator computed by comparing the current EPS to the one of the same period in the previous year. It measures the rate at which a company's EPS has increased over time and it is used to evaluate the future performance of a company. In particular, a high earnings growth rate indicates that a company is increasing fast its earnings therefore its market price will probably grow.

    5. Earnings Yield: it is a financial indicator that is the inverse of the PE ratio, so it is computed by dividing a company's EPS by its current market price per share. It measures the return on investment that a stock provides based on its EPS relative to its current market price and it is used to determine whether a stock is undervalued or overvalued. In particular, a high earnings yield indicates that a stock is providing a high return on an investment relative

to its market price and may represent an investment opportunity. Instead, a low earnings yield may indicate that a stock is not providing an attractive return on investment.

- Technical Indicators [38]: These indicators are daily technical indicators generated by mathematical calculations based on historical prices and/or volume data.

  1. Volatility: it is a financial indicator that measures the stability of a stock price, and the risk of an investment and also can evaluate the overall market conditions. For instance, high volatility can indicate that a stock is risky and may be subject to significant price fluctuations. We computed it as the 20-day exponentially weighted standard deviation of the stock performance relative to the market.

  2. Bollinger Bands: it is a technical indicator that represents a confidence interval for the stock price and can be used to measure volatility. It is constructed through two standard deviation above and below the 20-day moving average of the stock performance. The upper band represents an overbought condition, and the lower band represents an oversold condition. When the price of a stock moves outside of the upper or lower band, it indicates that the stock is becoming overbought or oversold, and a trend reversal may occur. Therefore, when the indicator falls below the lower band, it is recommended to buy since we expect the price to rise. Conversely, if the indicator rises above the upper band, it is advised to sell as a downward trend in the price is expected. As features, we consider the difference between the bands and the correspondent sector price.

  3. Relative Strength Index (RSI): it is a technical momentum indicator used to determine the overbought and oversold conditions of a stock by comparing the magnitude and the speed of recent price changes.

  4. Moving Average Convergence Divergence (MACD): it is a technical indicator computed as the difference between the 12-day exponential moving average and the 26-day exponential moving average of the asset's price. It is used to identify bullish or bearish market conditions. In particular, when the MACD is positive it is considered a bullish signal while when it is negative it is considered a bearish signal.

  5. Signal Line: it is a technical indicator computed as a moving average of another indicator in order to act as a signal for buy or sell. As a feature we consider it

combined with the Moving Average Convergence Divergence: in this case, the signal line is a 9-day exponential moving average of the MACD line. When the MACD line crosses above the signal line we have a bullish trend, while when the line crosses below the signal line we have a bearish trend. In the first case, we expect the price to rise, in the second case we expect the price to fall.

6. Diff: it is the difference between Macd and Signal Line. It can be used to identify the fluctuations and trends of stock prices in order to determine whether it is better to buy or sell. In particular, a transition from a negative to a positive value may be interpreted as a buy signal, while a transition from a positive to a negative value may be interpreted as a sell signal.

7. Volume: it refers to the number of shares or contracts traded in the financial market during a specified period of time. It is used to detect price movements and to confirm trends identified by other technical indicators. In particular, if a stock is trending upward and the volume is increasing, it may suggest that the trend is likely to continue. Conversely, if the stock is trending downward and volume is increasing, it may suggest that the trend is likely to reverse.

8. Volume Oscillator: it is a technical indicator used to measure the changes in trading volumes over a specified period of time. It is based on the difference between two moving averages (12-day and 26-day) of the volume.

The large number of financial indicators can lead to computational and sample complexities issues, as well as the possibility of having correlated features. Therefore a feature selection process will be conducted in the next chapter (5) to identify the most relevant and informative features. Starting from the results of this analysis, we define the state of our environment composed of 8 technical indicators (Bollinger Bands, diff, MACD, RSI, signal Line, volatility, Volume, Volume Oscillator) and the returns (daily, weekly, and weekly lagged). Specifically daily returns are the ones introduced in Equation 4.1. Similarly the weekly returns are computed considering a window of 5 days:

$$r_t = \frac{ClosePrice_t - ClosePrice_{t-5}}{ClosePrice_{t-5}} = \frac{ClosePrice_t}{ClosePrice_{t-5}} - 1 \tag{4.9}$$

The lagged returns we consider are the weekly returns of 2,3, and 4 previous weeks.

Additionally we consider the allocation at the previous step. This last feature is crucial because when the agent selects an action, he needs to take into account the previous position because of transaction costs. In particular, if the price changes by an amount

smaller than the transaction fee, the cost of the operation will be higher than the return, resulting in a negative reward. Adding the position of the previous step enables the agent to choose the 'hold' action if he predicts that the costs of a potential transaction exceed the returns. In other words, this feature is essential for helping the agent make more informed decisions on managing transaction costs and expected returns.

4.4. Algorithm Selection

In reinforcement learning, the *agent* is the entity that interacts with the environment to learn how to perform a task. The agent receives observations from the environment, takes actions based on its current policy, and receives rewards as feedback for its actions. The goal of the agent is to learn a policy that maximizes its cumulative reward over time.
In order to properly define our agent we opt for a value-based approach over a policy-based approach. While the latter is more sample efficient, it tends to be less stable and more sensitive to the quality of the function approximator and the choice of hyperparameters. In general, the main limitations of policy gradient algorithms are that they typically have a high number of hyperparameters that need to be tuned in order to achieve good performance. This can be time-consuming and can lead to instability issues. Moreover, they can get stuck in local optima, especially when dealing with large state and action spaces. This can result in suboptimal policies that are far from the true optimal policy. Moreover, a policy-based approach with a continuous action space can lead to dimensionality issues and consequently to suboptimal solutions.
On the other hand, the value-based approach is less sensitive to these factors and it is more stable, even though it may require a larger amount of data to converge to the optimal policy. Therefore, for our problem, it is better to adopt a value-based approach with action discretization.
We choose FQI algorithm because it is more robust than other value-based methods, such as DQN algorithm which is more sensitive to noise or inaccuracies in the data. Additionally, FQI is more interpretable since it uses XGBoost or Extra Trees instead of neural networks, which are a black box approach. Specifically Neural networks are often referred to as black box models because they are complex and difficult to understand. Unlike decision trees or XGBoost, which are based on a series of simple, interpretable rules that can be easily visualized and understood, neural networks consist of many interconnected layers of nodes that operate on input data in a highly nonlinear way. This complexity makes it difficult to understand how the network produces the predictions or decisions, which can be a disadvantage in some contexts where interpretability is important. As explained above, the purpose of this work is to be employed in a real-world financial con-

text, so it is essential to consider that the models need to be explained to investors. Therefore, we employ FQI algorithm to perform a weekly trading strategy for each sector individually. In order to train our model, we consider daily features as input.
To evaluate our model, we simulate the optimal trades according to the predictions of the model and compute the cumulative returns over the testing period.

4.4.1. FQI dataset

The dataset $\mathcal{D}$ required to train FQI algorithm consists of the four-values tuples $(s_t, a_t, s_{t+1}, r_{t+1})$ containing the previous state, the performed actions, the obtained reward, and the next state for all the possible combination of previous allocation and action:

$$\mathcal{D} = \{(s_t^k, a_t^k, s_{t+1}^k, r_{t+1}^k), k = 1, 2, ..., |\mathcal{D}|\} \tag{4.10}$$

5 | Data Analysis

In this chapter, we aim to find the most relevant features. We begin by analyzing the historical financial series relative to each asset to assess its predictive power. Then we apply effective techniques to remove non-stationarity.

5.1. Analysis of the Dataset

We focus on analyzing the sectors using the Global Industry Classification Standard (GICS, [7]). The sectors we consider are shown in Table 5.1.

After selecting the sectors, we need to rebuild their price series. In order to do that the natural choice is to consider the S&P 500 as a reference market index. S&P 500 index, in fact, represents the top 500 large-cap U.S. companies, and it is considered to be a leading indicator of U.S. equities, usually used as a benchmark for the overall performance of the U.S. stock market. Unfortunately, the stocks component, called *constituents*, of the S&P 500 index are not freely available through Bloomberg. As an alternative, we choose the B500 index, which is a Bloomberg proprietary index that replicates the S&P 500. Like the S&P 500, it is a stock market index, and it tracks the performance of the largest US companies but has the necessary data available. Therefore, using the constituents of the B500 index and their corresponding stock prices, we reconstruct the historical sectors price series (Figure 5.1).

Instead of only considering the sectors prices independently, we decide to use sector performance data relative to the market, which means that we construct an historical time series of sector minus market. In order to do this we compute the performance of each sector, and the performance of the B500 index and take the difference. We then recalculate the prices. By doing this, we are able to identify specific sector risks: by removing the market component, we eliminate the market risk, leading to a market neutral time series. We will refer to this new time series as *sectors vs market*.

Moreover, by removing market risk, we also obtain historical series with less noise, as we

GICS
Communication Services
Consumer Discretionary
Consumer Staples
Energy
Financial
Health Care
Industrial
Information Technology
Materials
Real Estate
Utilities

Table 5.1: Sectors from GICS classification

Fundamental Indicators
Price to Book (PB)
Price over Earnings (PE)
Dividend Yield
Earnings Growth
Earnings Yield

Table 5.2: Fundamental Indicators

can see in Figure 5.2.

Then we choose the most significant financial indicators, both fundamental and technical, as explained in Section 4.3. The fundamental indicators we chose are shown in Table 5.2. Like for sector prices, we consider these indicators relative to the market, constructing for each of them a new historical time series of *Indicators vs Market*. Similarly we construct the technical indicators (Table 5.3) in order to have all the variables relative to the market.

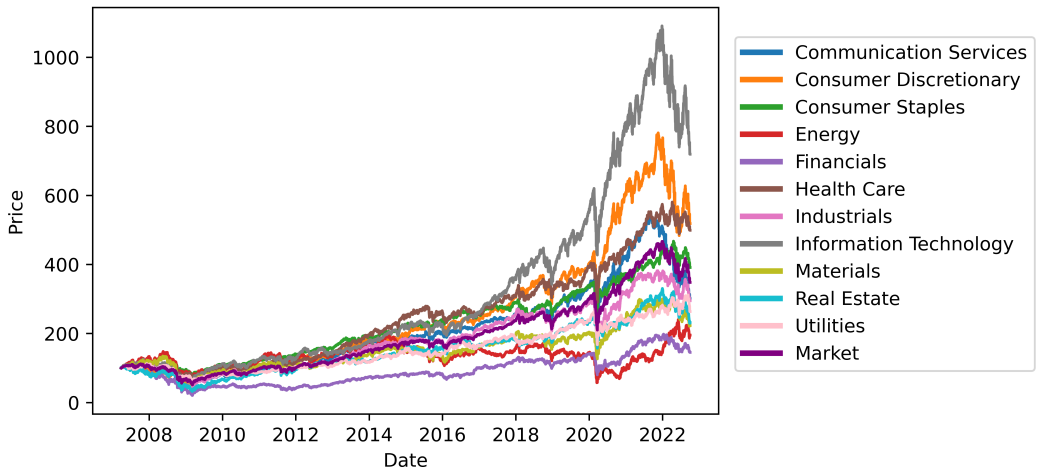

Figure 5.1: Historical Price Series - Sectors and Market

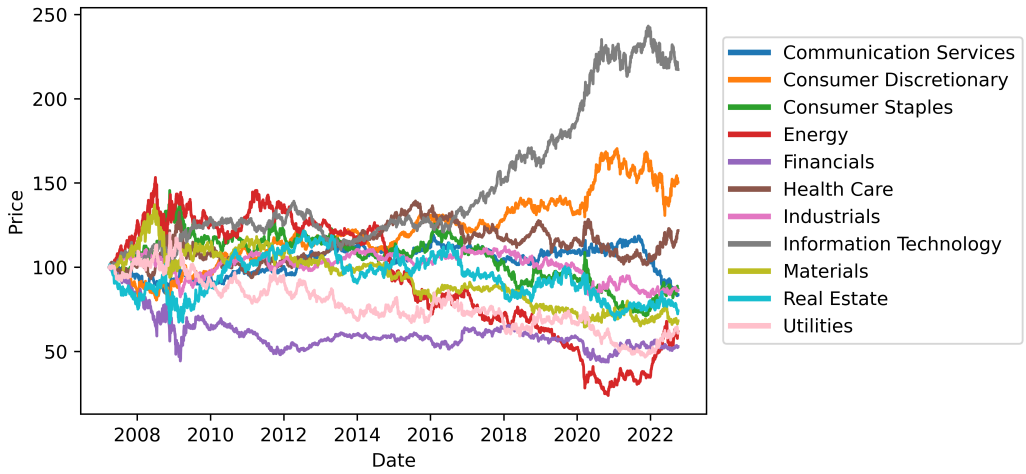

Figure 5.2: Historical Price Series - Sectors vs Market

Technical Indicators
Volatility
Bollinger Bands
Relative Strength Index (RSI)
Moving Average Convergence Divergence (MACD)
Signal Line
Volume
Volume Oscillator

Table 5.3: Technical Indicators

5.2. Features Selection

In this section, we perform a feature selection procedure to detect the most informative indicators, that is, which indicators have an effective predictive power on asset returns. In order to do that we choose ExtraTrees Regressor [22] and XGBoost [13], which are supervised learning algorithms that have been shown to be effective in producing accurate predictions for a wide range of regression problems [27, 37].

In a first instance, we have to introduce the concept of *Decision Tree*. A *Decision Tree* is a machine learning method that uses decision rules learned from data to predict the value of a target variable. The structure is composed of nodes and branches. At every node, the data is split based on a specific input feature, which generates multiple branches as output. This process continues with increasing numbers of branches until a node is formed where the data mostly belong to the same class, and no further splits or branches are possible. This results in a tree-like structure, with the first splitting node being the root node and the end nodes being the leaves.
There are two different types of decision trees:

- Classification Trees: These trees are used for predicting categorical or discrete values. They split the data based on the features. Each branch represents a decision rule based on a feature value, and each leaf node represents a class or category.
- Regression Trees: These trees are used for predicting continuous or numerical values. Like classification trees, they split the data based on the features, but instead of predicting a class, they predict the average value of the target variable for the data in each leaf node.

Extremely Randomized Decision Tree (Extra-tree) is a type of decision tree in which the cut-point while splitting a tree node, whereas decision trees choose the optimal split based on information gain or Gini impurity. This makes extra-trees more robust to noisy data and less prone to overfitting.

Extra Trees Regressor Extra Trees Regressor is a supervised learning method that creates many extra-trees to develop a more robust model. In particular, it provides an estimator that fits multiple extra-trees on various sub-samples of the dataset, followed by averaging the predictions to improve predictive accuracy and prevent over-fitting. The fact that the cut points are chosen randomly for each tree makes the trees diversified and uncorrelated.

XGBoost Extreme Gradient Boosting (XGBoost, [13]) is a decision-tree-based algorithm that combines multiple decision trees. It works in a gradient boosting framework, which is a technique that uses gradient descent to optimize the loss function of the model, providing a parallel tree boosting. It starts by fitting an initial predictor model (e.g. a tree) to the data. Then the algorithm works by iteratively adding predictors to the model while optimizing a given objective function. Each successive model attempts to correct for the shortcomings of the combined boosted ensemble of all previous models.

Extra Trees Regressor and XGBoost are both ensemble learning methods. XGBoost is a more powerful and efficient algorithm, but Extra Trees is simpler to use and can still achieve good performance. Moreover, Extra Trees has fewer parameters to be optimized therefore the tuning process is computationally less expensive.

As a final regression method, we decide also to consider the KNN Regressor [36], where KNN stands for K-Nearest Neighbors. It is denoted as a 'lazy algorithm', which means it doesn't build a model during training. Instead, it simply stores the training data and uses it during prediction. The algorithm first chooses a value for k, which will determine the number of nearest neighbors that will be used to predict the target variable. Then the distance between the new observation and each observation in the training set is computed. There are various methods for calculating this distance, of which the most commonly known methods are Euclidian, Manhattan, and Hamming distance ([58]). Finally, the algorithm selects the k nearest data points to the new observation, based on the calculated distances, and computes the average of the target variable for the k-nearest neighbors. The average value is the predicted value for the new observation.

In order to assess the accuracy of the forecast, commonly used statistics are the R^2 and the *Accuracy*.

Determination Coefficient R^2 is the coefficient of determination, which is a measure of how well the regression line fits the data. It ranges from 0 to 1, with higher values indicating a better fit. It is computed as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{5.1}$$

where n is the number of observations in the dataset, y_i is the observed value of the dependent variable for the ith observation, $\hat{y_i}$ is the predicted value of the dependent variable for the ith observation based on the model, $\bar{y}$ is the mean value of the dependent variable across all observations.

Instead of considering the R^2 for our analysis, we decide to consider adjusted R^2 statistic. It ranges from inf to 1,with higher values indicating a better fit. It can be expressed as:

$$R^2_{adj} = 1 - \frac{(1 - R^2)(n - 1)}{n - k - 1}, \tag{5.2}$$

where k is the number of features.

The adjusted R^2 is a modified version of the R^2 that takes into account the number of independent variables in the model. It adjusts the R^2 to better manage overfitting, which can occur when additional variables are added to the model that does not significantly improve its explanatory power. Like the R^2, higher values indicating a better fit.

Accuracy Accuracy is a commonly used metric to evaluate the performance of a classification model. It measures the proportion of correct predictions made by the model out of the total predictions made. In order to mathematically define the accuracy we have to introduce the *true/false positive/negative* measures. The *True Positives* (TP) refers to the number of positive instances that were correctly classified as positive by the model. *True Negatives* (TN) refers to the number of negative instances that were correctly classified as negative by the model. *False Positives* (FP) refers to the number of negative instances that were incorrectly classified as positive by the model. *False Negatives* (FN) refers to the number of positive instances that were incorrectly classified as negative by the model.
The accuracy is defined as the ratio of the number of correct predictions to the total number of predictions made by the model:

$$Accuracy = \frac{TN + TP}{TP + FP + TN + FN}, \tag{5.3}$$

In our case, since the return values are continuous, we have considered their sign to compute the accuracy. By doing this, we have a measure of how well the model can predict gains (positive returns) and losses (negative returns) correctly.

As statistics measures, we decide to consider both the adjusted R^2 and the *accuracy*. The adjusted R^2 is more suitable than accuracy because it takes into account not only the sign

Min Sample Split	R^2 Train	R^2 Test
0.01	0.618	-0.081
0.05	0.309	-0.031
0.1	0.243	-0.027
0.2	0.192	-0.025
0.7	0.127	-0.017
0.99	0.039	-0.005

Table 5.4: R^2 Score - Daily Analysis

of the prediction but also its magnitude. In fact in the case of a predicted negative return, we are interested in knowing how negative it is. In particular, the problem arises when the model predicts a negative return of a certain magnitude, but a much larger negative return occurs.
On the other hand the adjusted R^2 statistic may produce inaccurate or misleading results when dealing with datasets containing outliers or high variability. Therefore, when interpreting the adjusted R^2, we have to consider that the presence of outliers in the data can negatively effect the model's fit and predict. In this case *accuracy* can play a key role in order to understand the predictive power of the features in our model.

Results We first employ a daily approach, using daily data to predict daily returns. We divide the dataset into a training set and a test set, with proportions of 80% and 20%, respectively.
We present the results using Extra Tree Regressor: for every trial, we varied the number of trees in each forest ($n\_estimators$), the minimum number of samples required to split a node (min_sample_split) and the time window. We present the obtained adjusted R^2 for Consumer Discretionary sector with $n\_estimators = 500$, window $= 5$, as min_sample_split is varied in Table 5.4.

We also construct the *Confusion Matrix* considering the absolute value of the predicted returns in order to show the number of correct and incorrect sign predictions. Specifically the $ConfusionMatrix$ indicates how many TP, TN, FP, and FN are predicted. Therefore it is a way to visualize all the terms used to compute the *accuracy* (Formula 5.3). Results are shown in Figure 5.3 and 5.4 for $min\_sample\_split = 0.05$.

Unfortunately, we found no evidence of predictive power at the daily level. Therefore,

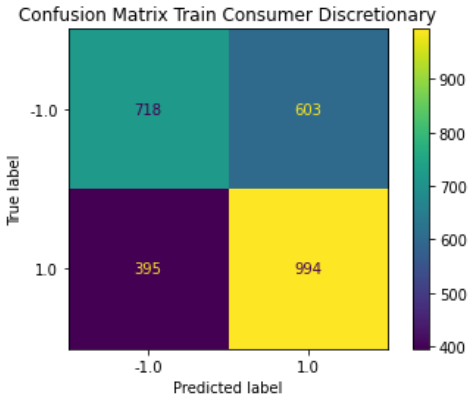

Figure 5.3: Confusion Matrix Train - Daily Analysis

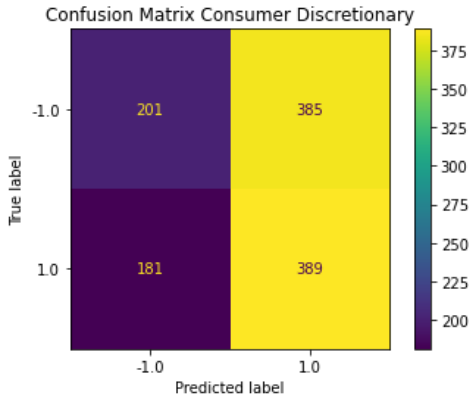

Figure 5.4: Confusion Matrix Test - Daily Analysis

Min Sample Split	R^2 **Train**	R^2 **Test**
0.01	0.654	-0.071
0.05	0.326	-0.026
0.1	0.283	-0.022
0.2	0.210	-0.019
0.7	0.138	-0.014
0.99	0.064	0.002

Table 5.5: R^2 Score - Weekly Analysis

we shifted our analysis to the weekly level using daily data to predict weekly returns. Although there was a slight improvement in the results in terms of adjusted R^2 (Table 5.5) and in terms of *accuracy* (Figure 5.5, Figure 5.6), the predictive power remained inadequate.

In light of this, we turned our attention to the impact of non-stationarity on the accuracy of the predictions. Additionally, the net division between train and test made it challenging to select the optimal regressor's parameters. To address these issues, we attempted to enhance the data stationarity and adopt a rolling approach with a validation set to optimize the parameter selection process.

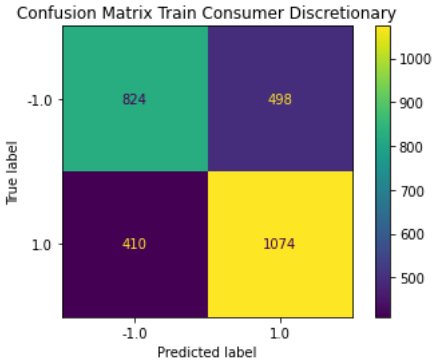

Figure 5.5: Confusion Matrix Train - Weekly Analysis

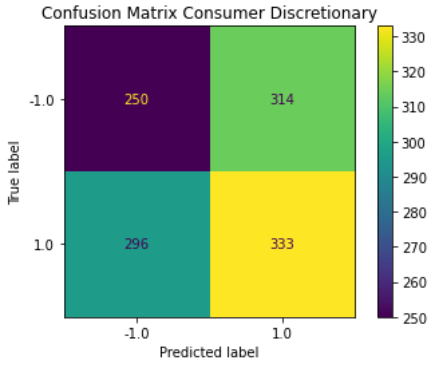

Figure 5.6: Confusion Matrix Test - Weekly Analysis

Min Sample Split	R^2 **Train**	R^2 **Test**
0.005	0.621	0.039
0.01	0.578	0.041
0.05	0.494	0.033
0.2	0.227	0.019

Table 5.6: R^2 Score - Shuffle

5.3. Stationarity

We randomize the historical time series in order to observe the impact of non-stationarity since shuffling data removes non-stationarity. Next, we split the data into a train set and a test set with a 20% test size, remove the test data from the training set, and calculate the adjusted R^2 score and the *accuracy* score while varying the min_sample_split to evaluate the predictive performance.
We present the results for the Consumer Discretionary sector using Extra Trees Regressor with a window of 5 days and 500 trees while varying the minimum number of data required to split a node. We obtain an improvement in the results in terms of adjusted R^2, as we can see in Table 5.6. Therefore we conclude that non-stationarity has a negative impact since the predictive power improves after removing it. Consequently, we attempt to make the data more stationary while preserving their predictive power and information.

To achieve this, we employ an approach called *Fractional Differentiation*, which was first introduced by Lopez de Prado in [17] and implemented on real data in [57]. *Fractional Differentiation* makes time series more stationary while retaining their memory and predictive power.

Year	p-value	Stationarity
2008	2.125e-2	stationary
2009	1.257e-3	stationary
2010	3.384e-2	stationary
2011	2.348e-4	stationary
2012	5.506e-4	stationary
2013	1.753e-3	stationary
2014	3.607e-6	stationary
2015	1.624e-3	stationary
2016	1.134e-2	stationary
2017	4.357e-2	stationary
2018	1.65e-5	stationary
2019	8.747e-7	stationary
2020	2.594e-4	stationary
2021	1.044e-6	stationary
2022	1.743e-5	stationary

Table 5.7: ADF Test - Bollinger Up

The most common method used to remove non-stationarity from data is to make the first (or in general some integer) order difference or the logarithm. However, these approaches erase a significant portion of the data memory, dramatically reducing their predictive power. Fractional differentiation overcomes this problem by finding the fraction d such that the data become sufficiently stationary while preserving as much information as possible. In particular, each past value of the time series is assigned a weight ω such that:

$$\omega = \{1, -d, \frac{d(d-1)}{2!}, ..., (-1)^k \prod_{i=0}^{k-1} \frac{d-i}{k!}\} \tag{5.4}$$

When d is equal to k, the memory beyond that point is removed. The goal is to find d such that stationarity is achieved and the maximum volume of memory of the time series is preserved.

We obtained good results in terms of stationarity by setting $d = 0.2$. In Table 5.7 we present the result of ADF test on the historical data of the technical indicator *Bollinger Up*. Similar results were obtained for the other features.

In particular we perform an augmented Dickey-Fuller test on each feature in the dataset after applying the *Fractional Differentiation* method for some value of d.

The Augmented Dickey-Fuller (ADF) test is a type of unit root test that examines how strongly a time series is defined by a trend. It does this by fitting an autoregressive model to the time series and optimizing an information criterion across multiple different lag values. The test then evaluates the null hypothesis that the time series can be represented by a unit root, indicating that it is non-stationary, against the alternative hypothesis that the time series is stationary. To interpret the results of the ADF test, we use the p-value obtained from the test. If the p-value is below a defined threshold α (set to 0.05), we reject the null hypothesis and conclude that the time series is stationary. Conversely, if the p-value is above the threshold, we fail to reject the null hypothesis and conclude that the time series is non-stationary.

Therefore, analyzing the p-value, we find that with d = 0.2 the data became stationary.

5.4. Rolling Approach

We decided to employ a rolling approach in order to avoid a net division between train and test set: considering a rolling window allows us to properly evaluate the predictive power of the features over different periods, leading to a more stable analysis. In order to find the best model parameters for each train-test period we used Optuna [2], which is a commonly employed open-source hyperparameter optimization framework for machine learning [29, 49].
We consider as training set 3 years of daily data and as validation period the following year. In order to find the best parameters we set the Optuna objective function as the R^2 computed on the weekly returns of the validation set. After identifying the optimal parameters, we train the model on 3 years of daily data to predict weekly returns for the following month. We then shift the dataset by one month and repeat this process. At the end of each year, i.e. after 12 shifts, we collect the predictions and evaluate their *accuracy* and adjusted R^2.
We present the results for sector Consumer Discretionary, considering Extra Trees Regressor. In Table 5.8, Figure 5.8 and Figure 5.9 we can see the results in terms of adjusted R^2 and *accuracy* with training set from 2018 to 2020 and test set 2021. Unfortunately, the results at the adjusted R^2 level are still negative, despite achieving a reasonable level of *accuracy*. In Table 5.9, Figure 5.10 and Figure 5.11 we can see the results in terms of adjusted R^2 and *accuracy* with training set from 2009 to 2011 and test set 2012. In this case we obtain both a negative adjusted R^2 and a low *accuracy*.

R^2 Train	0.68
R^2 Test	-0.03
Accuracy Train	0.87
Accuracy Test	0.56

Table 5.8: R^2 and *Accuracy* - Rolling Approach 2018-2021

R^2 Train	0.59
R^2 Test	-0.04
Accuracy Train	0.67
Accuracy Test	0.51

Table 5.9: R^2 and *Accuracy* - Rolling Approach 2009-2012

Through various years of train-test, we have always obtained negative R^2 values. However, our accuracy score has yielded some good results (>55%).

One of the reasons we identified for this behavior is that the adjusted R^2 may be too sensitive as a measure to evaluate prediction in such a volatile context, and already achieving discrete *accuracy* results (5.9) may indicate that some features are more significant than others.
Therefore, we have considered the feature importance results obtained from Extra Tree Regressor, which revealed that fundamental indicators are not relevant (Figure 5.7).
The importance of a feature derived from Extra Trees is computed using the Mean Decrease Impurity (MDI, [32]) method. This method calculates the total amount that the impurity of the target variable is decreased by splits over a particular feature.
For each tree the importance ni_j of node j is computed as:

$$ni_j = \omega_j C_j - \omega_{left(j)} C_{left(j)} - \omega_{right(j)} C_{right(j)} \tag{5.5}$$

where ω_j is the weighted number of sample reaching node j, C_j is the impurity value of node j, *left(j)* is the child node from left split on node j, *right(j)* is the child node from right split on node j.

The importance F_i of feature i on each extra tree is then calculated as:

$$F_i = \frac{\sum_{j \in \mathcal{F}_\rangle} ni_j}{\sum_{j \in \mathcal{N}} ni_j} \tag{5.6}$$

where $\mathcal{N}$ is the set of all the nodes and $\mathcal{F}_\rangle$ is the set of nodes that split in features i.
Then a normalization procedure is performed:

$$F_{i,norm} = \frac{F_i}{\sum_{j \in \mathcal{F}} F_i} \tag{5.7}$$

The final feature importance is the average over all the trees:

$$F_j = \frac{\sum_{t \in \mathcal{T}} F_{j,t}}{T} \tag{5.8}$$

where T is the total number of trees and $\mathcal{T}$ is the set of all trees.
The impurity metric C used in Extra Trees Regressor to calculate feature importance is the Gini impurity [61]. The Gini impurity is a measure of how often a randomly chosen element in a set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset. It is computed as:

$$G = \sum_{i=1}^{C} f_i(1 - f_i), \tag{5.9}$$

where f_i is the frequency of label i at a node and C is the number of unique labels.

Therefore, as a consequence of feature importance results, the set of features selected to perform further analysis includes only technical indicators and returns.
This result is quite intuitive even at a qualitative level since fundamental indicators depend heavily on prices (and therefore returns). For example, the PE ratio is the price divided by earnings, and earnings data are updated once every 3 months, while the price changes daily. Therefore, the daily and weekly impact depends only on prices.

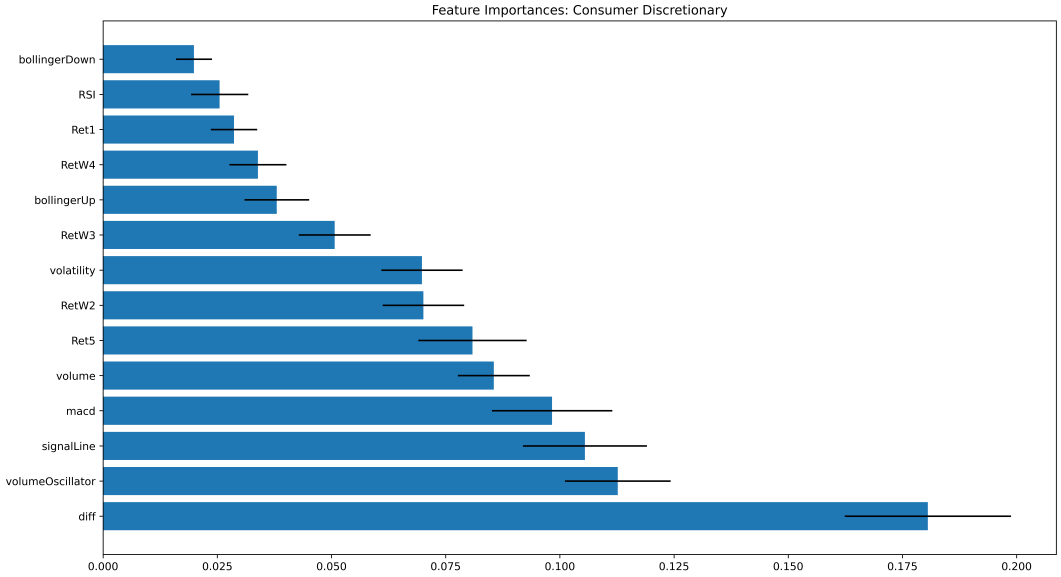

Figure 5.7: Feature Importance

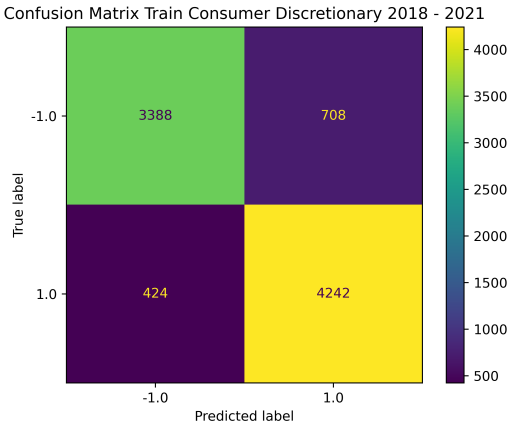

Figure 5.8: Confusion Matrix Train - Rolling Approach 2018-2021

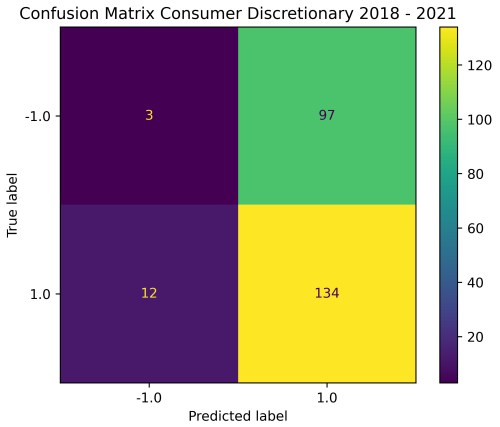

Figure 5.9: Confusion Matrix Test - Rolling Approach 2018-2021

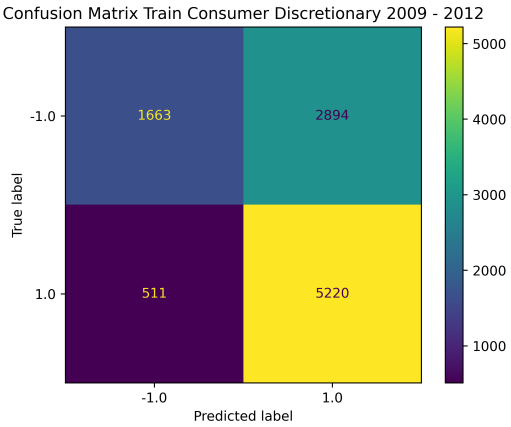

Figure 5.10: Confusion Matrix Train - Rolling Approach 2009-2012

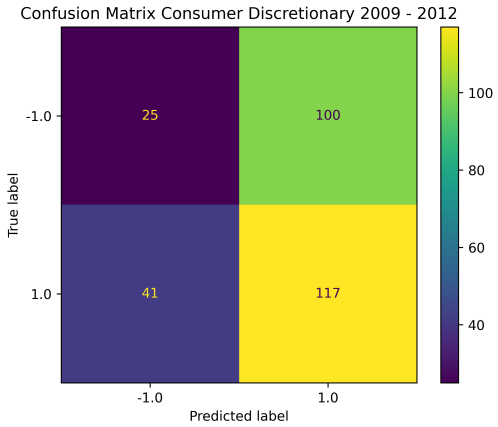

Figure 5.11: Confusion Matrix Test - Rolling Approach 2009-2012

5.5. Recursive Feature Addition

Recursive feature addition (RFA, [20]) is a machine learning method that performs feature selection by iteratively adding features to a model and evaluating their importance until a desired level of accuracy is achieved. The process starts with an empty set of features and the model is trained and evaluated using only a single feature. The feature with the highest importance score is then added to the feature set, and the model is trained and evaluated again using this new set of features. This process is repeated, with the next highest importance feature being added at each iteration until the desired level of accuracy is achieved or all features have been added.

The advantage of RFA over other feature selection methods is that it takes into account the interaction between features and their combined importance to the model. However, it can be computationally expensive, especially for large datasets with many features.
In algorithm 5.1 we present the RFA algorithm.

In particular, VR is a variable ranking algorithm and MB is the model-building approach. The former is used to determine the most important variable based on a specified criterion. The latter is used to build the regressor model and estimate the variable of interest. It takes as input the set of selected features from the VR algorithm and uses them to train the model. The MB approach then estimates the variable of interest by making predictions using the trained model. By iteratively adding features based on their importance scores and building the model, RFA gradually selects the most relevant set of features for accurately predicting the variable of interest.
We employ Extra Trees Regressor and rank the features according to their importance derived from the estimator, as explained in Section 5.4. Our approach involves conducting training and validation to identify optimal parameters that would maximize the adjusted R^2 on the validation set. Once we had fine-tuned the parameters, we evaluated the performance of the model on the test set considering only features selected by the RFA. Unfortunately, even in this case, we obtain a negative adjusted R^2, equal to -0.02.

Algorithm 5.1 Recursive Feature Addition

Input: A dataset D, the variable to be explained V^o
Output: V_{sel}: a set of variables selected to estimate V^o
Initialize: $V_{sel} \leftarrow \emptyset$,$\hat{V}^o \leftarrow V^o$, $R^2_{old} \leftarrow 0$
while $\Delta R^2 > \epsilon$ **do**
  $V^\star \leftarrow \arg\max_{V \in D} VR(D, \hat{V}^o, V)$
  if $V^\star \in V_{sel}$ **then**
    return V_{sel}
  end if
  $V_{sel} \leftarrow V_{sel} \cup V^\star$
  $\hat{f} \leftarrow MB(V_{sel}, V_o)$
  $V \leftarrow V \setminus \hat{f}(V_{sel})$
  $\Delta R^2 \leftarrow R^2(D, V^o, \hat{V}^o) - R^2_{old}$
  $R^2_{old} \leftarrow R^2(D, V^o, \hat{V}^o)$
end while

6 | Experimental evaluation

In this chapter, we present the results achieved by training the FQI model on real market data. Specifically we proceed to train the model on daily data in order to construct an optimal weekly trading strategy for every asset. Finally, we evaluate the model performance on out-of-sample data.

6.1. Backtest

We trained the FQI model during the period from 2008 to 2015, using the 2016-2019 data as a validation set to obtain optimal parameters for the FQI regressor through Optuna Optimizator. We experiment XGBoost and Extra Trees as regressors, with XGBoost being selected due to its superior performance in computational efficiency and accuracy. To ensure the robustness of the optimal parameters, the Optuna objective function is set to maximize the average cumulative reward during the validation period across three different independent runs.
Finally, the validation set is included again in the training period and the model is retrained using the previously obtained parameters on daily data from 2008 to 2019 (Figure 6.2), with performance evaluation conducted on weekly out-of-sample data from 2020 to 2022 with 5 FQI iterations. We assume transaction costs of 0.0005 for each operation. Therefore, in our problem, we have to consider for each transaction a cost of 0.001 as our historical series are *sector vs market*, which requires two transactions each time a signal is generated. In fact when we have, for instance, a buy signal, it results in double transaction costs since we have to buy the sector index and sell the market.
To test the stability and robustness of the model, we perform backtesting simulations for the same sector using various seeds to observe the strategy volatility.

We present the results for the sector Consumer Discretionary, whose price (against the market) is shown in Figure 6.1. We consider a 5-day window and XGboost as regressor. Similar results were obtained for other sectors.

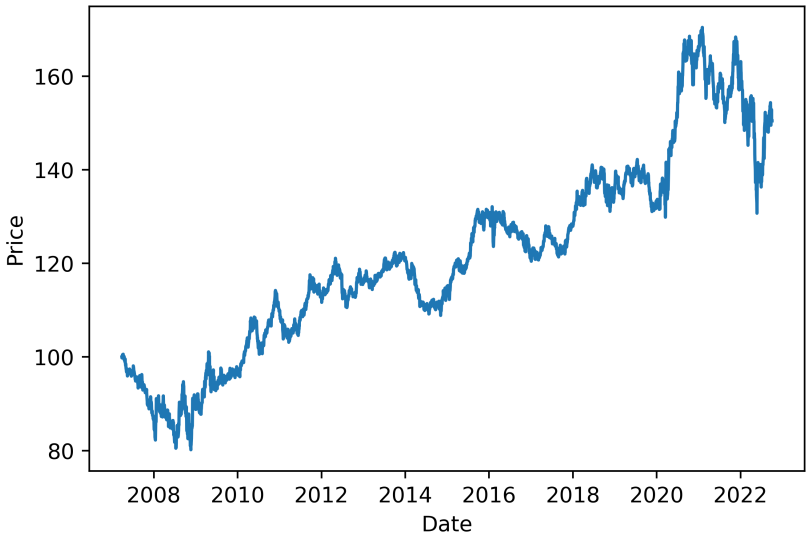

Figure 6.1: Consumer Discretionary vs Market

We obtain positive performance in backtesting for certain iterations of FQI, such as the 3th iteration which yielded a mean return of +7.8% across various seeds (Figure 6.3). However, the results from validation set (Figure 6.4) indicates that the iteration to choose in the test is the 1th, since it is the iteration that leads to the better performance (+10.6%). Therefore if we consider the first iteration of FQI for the test we get -5.3% (Figure 6.5). One reason we identified for this negative results it that the choice of the iteration to consider is biased due to our use of Optuna which optimizes the parameters only in the first iteration of FQI. Specifically we train the model and we employ Optuna to find the parameters that maximize the cumulative return on the validation set, but only for the first iteration of FQI. This is because the Optuna research is computationally really expensive therefore the optimization procedure is limited to the first iteration. However selecting the best parameters for each iteration during the validation procedure would lead to a more stable and correct choice of what FQI iteration to consider, instead of rely only on the optimal parameters of the initial iteration.
However, concerns regarding the volatility and instability of the resulting strategies persist, especially when we compare the results obtained with different independent runs (Figures 6.2, 6.4 and 6.3).
In conclusion, while the tested FQI iterations have shown promising performance for some US sectors, further investigation is needed to address the identified issues and ensure the reliability of the proposed trading model.

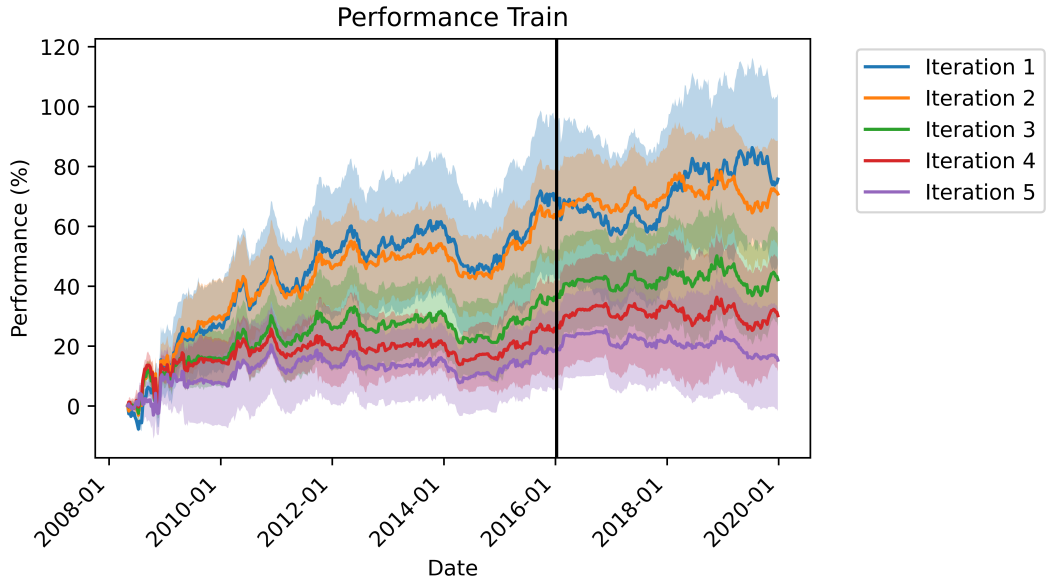

Figure 6.2: Performance Train - Consumer Discretionary

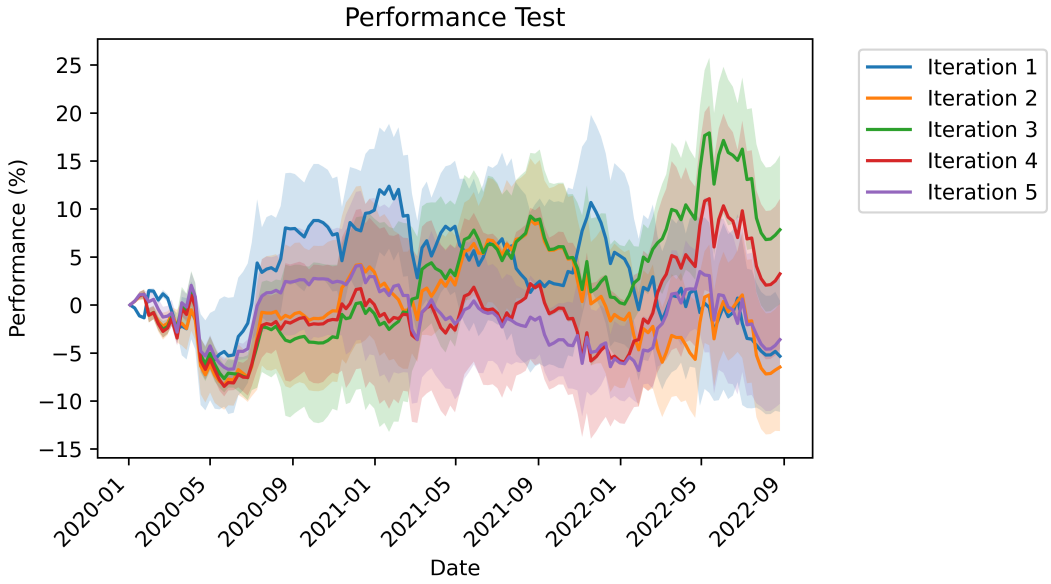

Figure 6.3: Performance Test - Consumer Discretionary

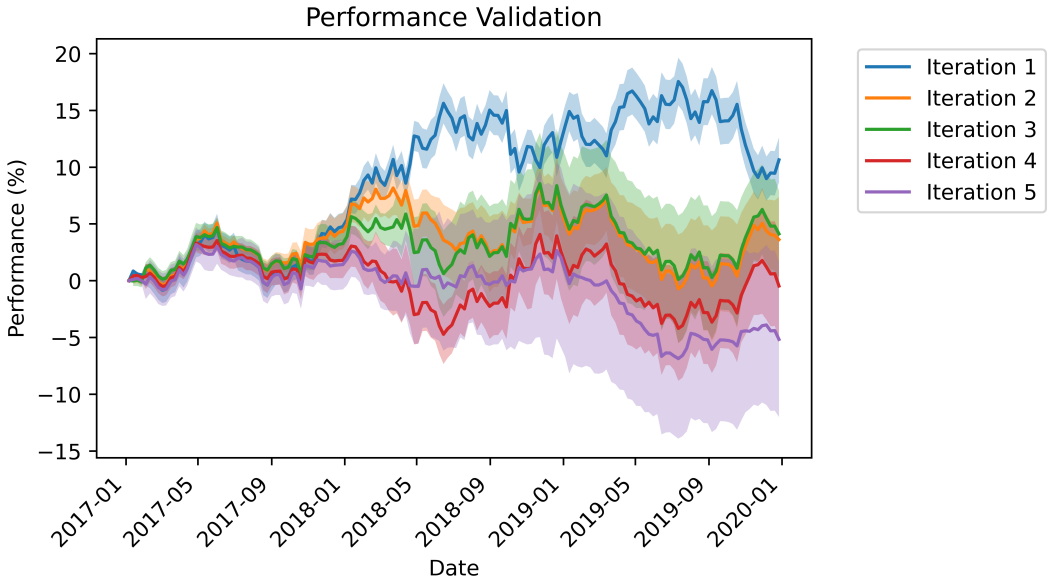

Figure 6.4: Performance Validation - Consumer Discretionary

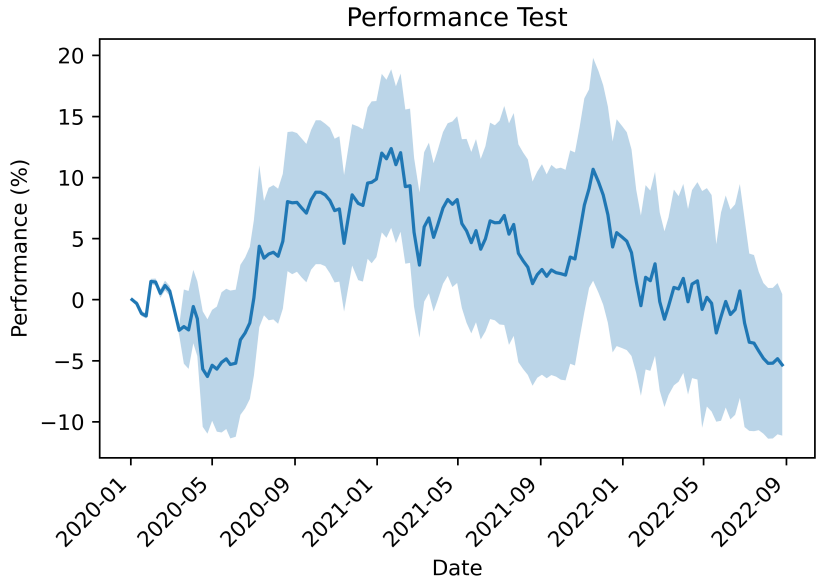

Figure 6.5: Performance Test, First Iteration - Consumer Discretionary

7 | Conclusions and future developments

The aim of this work is to use reinforcement learning to construct a trading model for US market sector indices. We begin by selecting and analyzing a set of financial indicators in order to identify the most relevant ones in terms of predictive power on asset returns. We consider both fundamental and technical financial indicators. We employ the *Fractional Differentiation* approach to remove non-stationarity in the data while preserving their predictive power. In order to evaluate the predictive power of our features we decide to employ a rolling approach with Extra Trees Regressor. Although the adjusted R^2 statistic did not indicate a strong predictive power among the features, we obtained a discrete level of accuracy. Therefore we selected the most relevant features for further analysis according to the Extra Trees feature importance.
Finally we trained the FQI algorithm on the selected features. The choice of FQI was made as it offers a transparent decision-making process, thereby avoiding black box issues. However, although the robustness of the algorithm, we notice that the resulting strategies exhibit high volatility and instability issues. Therefore there are several future developments that could enhance this work, including:

- investigating additional financial indicators in order to find other significant features which can improve the predictive power on asset returns;
- exploring alternative approaches for managing the non-stationarity of data;
- exploring alternative approaches to perform feature selection, e.g. discretize the features using quantiles and employ a classifier rather than a regressor. This approach makes the regressor more stable and efficient: grouping the features in classes instead of considering continue values lead to a less complex models.
- developing effective techniques to manage the volatility, e.g. set a volatility target and normalize the historical time series of the sectors and the market by this target. This approach makes the historical time series more stable.

In conclusion, we developed an effective trading model with the final goal of constructing a market-neutral portfolio based on the strategies obtained with the developed approach. One way to do this can be to consider the Q-value Function of the FQI agent's buy action for each sector to derive the optimal weights since assets with lower Q-values should have lower weights. The Q-value, in fact, represents the expected return when buying the asset, enabling us to assign lower weights to sectors with lower expected returns. Additionally, we must perform volatility controls to penalize assets with high expected returns that also have higher volatility.

This approach can be extended to any investment universe, not just sector indices but also other real asset classes such as equity indices, multi-asset portfolios or synthetic asset classes such as multi-factor portfolios or ETFs.

Bibliography

[1] A. M. Aboussalah and C.-G. Lee. Continuous control with stacked deep dynamic recurrent reinforcement learning for portfolio optimization. *Expert Systems with Applications*, 140:112891, 2020.

[2] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[3] H. Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

[4] R. J. Bauer and J. R. Dahlquist. *Technical Markets Indicators: Analysis & Performance*, volume 64. John Wiley & Sons, 1998.

[5] J. Baz and H. Guo. An asset allocation primer: connecting markowitz, kelly and risk parity. *PIMCO Quantitative Research*, 2017.

[6] E. Benhamou, D. Saltiel, J. J. Ohana, J. Atif, and R. Laraki. Deep reinforcement learning (drl) for portfolio allocation. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part V*, pages 527–531. Springer, 2021.

[7] S. Bhojraj, C. M. Lee, and D. K. Oler. What's my line? a comparison of industry classification schemes for capital market research. *Journal of accounting research*, 41 (5):745–774, 2003.

[8] L. Bisi, P. Liotet, L. Sabbioni, G. Reho, N. Montali, M. Restelli, and C. Corno. Foreign exchange trading: A risk-averse batch reinforcement learning approach. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020.

[9] L. Bisi, L. Sabbioni, E. Vittori, M. Papini, and M. Restelli. Risk-averse trust region optimization for reward-volatility reduction. In *Proceedings of the Twenty-Ninth In-*

ternational Conference on International Joint Conferences on Artificial Intelligence, pages 4583–4589, 2021.

[10] V. Carlini. L'intelligenza artificiale aiuta gli investimenti, ma serve più conoscenza. `https://www.quotidiano.ilsole24ore.com/sfoglio/aviator.php?newspaper=S24&edition=SOLE&issue=20230311&startpage=2&displaypages=2&articleId=1867716`, 2023.

[11] A. Castelletti, S. Galelli, M. Restelli, and R. Soncini-Sessa. Tree-based variable selection for dimensionality reduction of large-scale control systems. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 62–69. IEEE, 2011.

[12] Y. Chan, K. Hogan, K. Schwaiger, and A. Ang. Esg in factors. *The Journal of Impact and ESG Investing*, 1(1):26–45, 2020.

[13] 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*, pages 785–794, 2016.

[14] W. Chen, H. Zhang, M. K. Mehlawat, and L. Jia. Mean–variance portfolio optimization using machine learning-based stock price prediction. *Applied Soft Computing*, 100:106943, 2021.

[15] P. Cunningham, M. Cord, and S. J. Delany. Supervised learning. *Machine learning techniques for multimedia: case studies on organization and retrieval*, pages 21–49, 2008.

[16] R. Dash and P. K. Dash. A hybrid stock trading framework integrating technical analysis with machine learning techniques. *The Journal of Finance and Data Science*, 2(1):42–57, 2016.

[17] M. L. de Prado. Advances in financial machine learning, 2018.

[18] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[19] A. Edwards. Relational agency: Learning to be a resourceful practitioner. *International journal of educational research*, 43(3):168–182, 2005.

[20] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6, 2005.

[21] R. A. Fisher. *Statistical methods for research workers*. Springer, 1992.

[22] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63:3–42, 2006.

[23] Y. Guo, X. Fu, Y. Shi, and M. Liu. Robust log-optimal strategy with reinforcement learning. *arXiv preprint arXiv:1805.00205*, 2018.

[24] Z. Jiang, D. Xu, and J. Liang. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*, 2017.

[25] O. Jin and H. El-Saawy. Portfolio management using reinforcement learning. *Stanford University*, 2016.

[26] N. Justesen, P. Bontrager, J. Togelius, and S. Risi. Deep learning for video game playing. *IEEE Transactions on Games*, 12(1):1–20, 2019.

[27] E. Karbassiyazdi, F. Fattahi, N. Yousefi, A. Tahmassebi, A. A. Taromi, J. Z. Manzari, A. H. Gandomi, A. Altaee, and A. Razmjou. Xgboost model as an efficient machine learning approach for pfas removal: Effects of material characteristics and operation conditions. *Environmental Research*, 215:114286, 2022.

[28] L. Kennedy-Shaffer. Before p$<$ 0.05 to beyond p$<$ 0.05: using history to contextualize p-values and significance testing. *The American Statistician*, 73(sup1):82–90, 2019.

[29] J.-P. Lai, Y.-L. Lin, H.-C. Lin, C.-Y. Shih, Y.-P. Wang, and P.-F. Pai. Tree-based machine learning models with optuna in predicting impedance values for circuit analysis. *Micromachines*, 14(2):265, 2023.

[30] Z. Liang, H. Chen, J. Zhu, K. Jiang, and Y. Li. Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940*, 2018.

[31] Y.-C. Lin, C.-T. Chen, C.-Y. Sang, and S.-H. Huang. Multiagent-based deep reinforcement learning for risk-shifting portfolio management. *Applied Soft Computing*, 123:108894, 2022.

[32] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts. Understanding variable importances in forests of randomized trees. *Advances in neural information processing systems*, 26, 2013.

[33] Y. Ma, R. Han, and W. Wang. Portfolio optimization with return prediction using deep learning and machine learning. *Expert Systems with Applications*, 165:113973, 2021.

[34] H. Markowitz. Portfolio selection, the journal of finance. 7 (1), 1952.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[36] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[37] S. R. Polamuri, K. Srinivas, and A. K. Mohan. Stock market prices prediction using random forest and extra tree regression. *Int. J. Recent Technol. Eng*, 8(1):1224–1228, 2019.

[38] R. Pramudya and S. Ichsani. Efficiency of technical analysis for the stock trading. *International Journal of Finance & Banking Studies*, 9(1):58–67, 2020.

[39] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[40] A. Riva, L. Bisi, P. Liotet, L. Sabbioni, E. Vittori, M. Pinciroli, M. Trapletti, and M. Restelli. Learning fx trading strategies with fqi and persistent actions. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–9, 2021.

[41] T. Roncalli and G. Weisang. Risk parity portfolios with risk factors. *Quantitative Finance*, 16(3):377–388, 2016.

[42] Y. Sato. Model-free reinforcement learning for financial portfolios: a brief survey. *arXiv preprint arXiv:1904.04973*, 2019.

[43] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[44] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[46] W. F. Sharpe. The sharpe ratio. *Streetwise–the Best of the Journal of Portfolio Management*, 3:169–185, 1998.

[47] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[48] F. Soleymani and E. Paquet. Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder—deepbreath. *Expert Systems with Applications*, 156:113456, 2020.

[49] P. Srinivas and R. Katarya. hyoptxg: Optuna hyper-parameter optimization framework for predicting cardiovascular disease using xgboost. *Biomedical Signal Processing and Control*, 73:103456, 2022.

[50] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2018.

[51] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[52] C. Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[53] M. Taghian, A. Asadi, and R. Safabakhsh. Learning financial asset-specific trading rules via deep reinforcement learning. *Expert Systems with Applications*, 195:116523, 2022.

[54] E. O. Thorp. Portfolio choice and the kelly criterion. In *Stochastic optimization models in finance*, pages 599–619. Elsevier, 1975.

[55] M. Tran, D. Pham-Hi, and M. Bui. Optimizing automated trading systems with deep reinforcement learning. *Algorithms*, 16(1):23, 2023.

[56] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[57] R. Walasek and J. Gajda. Fractional differentiation and its use in machine learning. *International Journal of Advances in Engineering Sciences and Applied Mathematics*, 13(2-3):270–277, 2021.

[58] J. Walters-Williams and Y. Li. Comparative study of distance functions for nearest neighbors. In *Advanced techniques in computing sciences and software engineering*, pages 79–84. Springer, 2010.

[59] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[60] L. Weijs. Reinforcement learning in portfolio management and its interpretation. *Erasmus Universiteit Rotterdam*, 81:82, 2018.

[61] Y. Yuan, L. Wu, and X. Zhang. Gini-impurity index analysis. *IEEE Transactions on Information Forensics and Security*, 16:3154–3169, 2021.

List of Figures

List of Tables

Acknowledgements

Ringrazio il prof. Marcello Restelli senza il quale questa tesi non sarebbe stata possibile.
Ringrazio Luca e Antonio per avermi seguito e aiutato con grande dedizione in questo percorso.
Ringrazio Marcello Becchi per la sua costante disponibilità e pazienza non solo nella stesura di questa tesi ma anche nel trasmettermi le conoscenze necessarie affinchè potessi inserirmi al meglio nel mondo del lavoro.
Ringrazio Edwing per essermi stato sempre vicino nonostante la distanza, per avermi insegnato che non è necessario essere vicini di banco per essere sempre presenti e partecipi nella vita altrui. Grazie anche per il supporto informatico.
Ringrazio Pes e Cate per essere stati i migliori compagni di università, di studio e di vita che potessi incontrare. Siete sempre riusciti a trasformare le sventure in allegri momenti in cui lamentarci insieme, e allo stesso tempo a celebrare con sincera felicità i successi di ognuno di noi. La signora in giallo ha rappresentato molto durante questo percorso e per questo vi sarò eternamente grata.
Ringrazio Gianmarco per tutto l'affetto e per aver sempre creduto in me. Grazie della sincera stima. Averti a fianco in questo percorso mi ha arricchito non solo a livello accademico (grazie per Python) ma soprattutto a livello personale. Averti a fianco in questi anni mi ha reso una persona migliore.
E infine ringrazio i miei genitori, per tutto l'amore, il supporto costante, per le telefonate giornaliere. Grazie per avermi fatto sempre sentire amata. Grazie per esserci sempre stati, sia nei momenti belli sia in quelli difficili. Grazie per avermi insegnato che solo con la fatica e il duro lavoro si raggiungono importanti risultati. Grazie per avermi insegnato che studiare mi avrebbe reso una persona libera.