University of Padova

Department of Physics and Astronomy "Galileo Galilei"

Master Thesis in Physics of Data

Study and Implementation of Quantum-inspired Boosting Algorithms for A.I. powered Financial Asset Management

Supervisor
Prof. Marco Baiesi
University of Padova

External supervisor
Giovanni Davoli
Axyon AI

Master Candidate
Alessandro Lambertini

Student ID
1242885

Academic Year
2022-2023

"Tortoise: I don't see why you call them "defective". It is simply an inherent fact about record players that they can't do all that you might wish them to be able to do. But if there is a defect anywhere, is not in THEM, but in your expectations of what they should be able to do! And the Crab was just full of such unrealistic expectations"
— Contracrostipunctus - G.E.B. - Douglas Hofstadter

Abstract

Ensemble Learning (EL) is a machine learning technique that involves combining multiple individual models, called weak learners, in order to produce more accurate predictions. The idea behind EL is that by aggregating the predictions of multiple models, the final prediction can be more robust, accurate, and generalizable than that of any of the single weak learners alone.

Boosting is a powerful EL method in which the ensemble of models is constructed iteratively, so that at each iteration the training of new learners focuses on the training examples for which the previously selected models perform poorly.

Boosting algorithms have been successfully applied to various domains, including image and object recognition, text mining, finance and a number of other fields. They are particularly effective in scenarios where high accuracy and stability are crucial, making them a valuable tool in the field of machine learning.

Qboost is a boosting algorithm first introduced by Neven et al. in 2008 that casts the problem of EL into a hard combinatorial optimization problem that takes the form of a QUBO (Quadratic Unconstrained Binary Optimization) problem or, equivalently, an Ising model optimization.

Instances of this class of problems can be NP-complete and therefore difficult to tackle with classical digital computing methods and algorithms like simulated annealing (SA). Hence, alternative computational methods like the ones developed within the framework of quantum computing are of high interest for this class of problems. In particular, adiabatic quantum annealing (AQA) has been recently used for multiple demonstrations in the fields of particle detection, aerial imaging and financial applications. Its implementation on neutral atom processors, a type of adiabatic quantum hardware, has yielded promising results in terms of practical usefulness and scalability.

This thesis aims to develop, test and benchmark a Qboost-based algorithm in the context of multi-label classification problems. The project matured during an internship experience at Axyon AI, a FinTech company that serves quantitative asset managers through its proprietary machine learning software platform. The research and implementation showcased in this work serve as an initial step for a broader project designed to incorporate quantum-based algorithms and computational resources into Axyon AI's technological stack.

Axyon AI exploits EL and boosting in its machine learning pipeline. The scope of this project is to build a proof of concept for the improvement of the performance of the ensemble building step in the pipeline with respect to the currently employed EL algorithm.

The proposed techniques facilitate a broader exploration of the configuration space of the candidate models for the ensemble formation. It is hypothesized that this approach may potentially lead to maximizing performance and capturing untapped potential.

The outcomes of implementing and evaluating the new algorithms indicate that there are opportunities for enhancing the ensemble building process, as compared to the existing approach at Axyon AI. Specifically, a refined version of Axyon's current ensembling algorithm, which was crafted to serve as a performance benchmark against the Qboost-based approach, and the Qboost-based model itself, demonstrate improvements. These were not just in terms of overall performance but also in mitigating overfitting.

Contents

Listing of figures

Listing of tables

List of Algorithms

Listings

Listing of acronyms

ML ..... Machine Learning
EL ..... Ensemble Learning
DL ..... Deep Learning
DEL ..... Deep Ensemble Learning
SVM ..... Support Vector Machine
k-NN ..... k-Nearest Neighbors
QUBO ..... Quadratic Unconstrained Binary Optimization
SA ..... Simulated Annealing
QA ..... Quantum Annealing
AQA ..... Adiabatic Quantum Annealing
AT ..... Adiabatic Theorem
LtR ..... Learning to Rank
OOS ..... Out-of-sample

1 Introduction

This chapter outlines the fundamental concepts and methodologies that serve as the basis for the research conducted in this thesis. Initially, the essential definitions and techniques in the field of ensemble learning are presented. This is followed by an introduction to Quadratic Unconstrained Binary Optimization (QUBO) problems, along with a discussion of relevant optimization methods. The chapter concludes with an overview of the original Qboost algorithm, as described in Neven et al. (2008) [2], which forms the foundation for the present study.

1.1 Ensemble Learning

The expression "Ensemble Learning" (EL) refers to the machine learning (ML) framework where multiple models, often termed "weak learners" or "base learners" are trained to solve the same problem and their predictions are aggregated to form a single, unified prediction that is the expression of a "strong learner". The genesis of EL techniques can be traced back around the '90 when several foundational works have been published introducing some fundamental methods in this field like Bagging [3], Boosting [4] and Stacking [5] . Over the years, EL has proven to be remarkably effective in various applications ranging from natural language processing [6] to financial forecasting [7]. Its robustness in handling noisy, imbalanced, and incomplete data sets gives it an edge over single-model approaches. Moreover, by aggregating predictions from multiple models, EL naturally embodies a form of regularization, reducing the risk of overfitting.

1.1.1 Deep Ensemble Learning

Deep Ensemble Learning (DEL) is an extension of traditional Ensemble Learning, which incorporates Deep Learning (DL) models as base learners. DL models have shown exceptional predictive accuracy across various applications, such as computer vision and natural language processing.

One of the challenges of DEL is its computational intensity. Training multiple DL models, each with potentially millions of parameters, requires significant computational resources. This is in contrast to traditional ensemble methods that use simpler models like Decision Tree Classifiers, Support Vector Machines (SVM) , or k-Nearest Neighbors (k-NN), which are less computationally demanding.

Furthermore, deep learning models tend to be less diverse compared to simpler models. This lack of diversity can affect the ensemble's performance, as ensembles benefit from diversity among the base learners [8].

1.1.2 Mathematical Framework

Consider a dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_S, y_S)\}$, where S is the size of the dataset, x represents the feature vectors and y represents the corresponding labels or outcomes. Each weak learner h_i produces a hypothesis $h_i : \mathbb{R}^m \to \mathbb{Y}$, where $\mathbb{Y}$ is the output space (either $\mathbb{R}$ for regression or $\{0, 1\}$ for classification).

The ensemble learning framework utilizes an aggregation function φ to combine these multiple hypotheses into a single, strong hypothesis H, as depicted in Equation (1.1):

$$H(x) = \varphi(h_1(x), h_2(x), \ldots, h_N(x)) \tag{1.1}$$

1.1.3 Construction of Weak Learners

There are primarily two dimensions to consider when constructing the set $\{h_i\}$ of weak learners.

Architecture of Learners

- **Homogeneous EL**: All base learners share the same architecture. They differ only in the parameters obtained after training.
- **Heterogeneous EL**: The ensemble consists of a diverse set of models with different architectures.

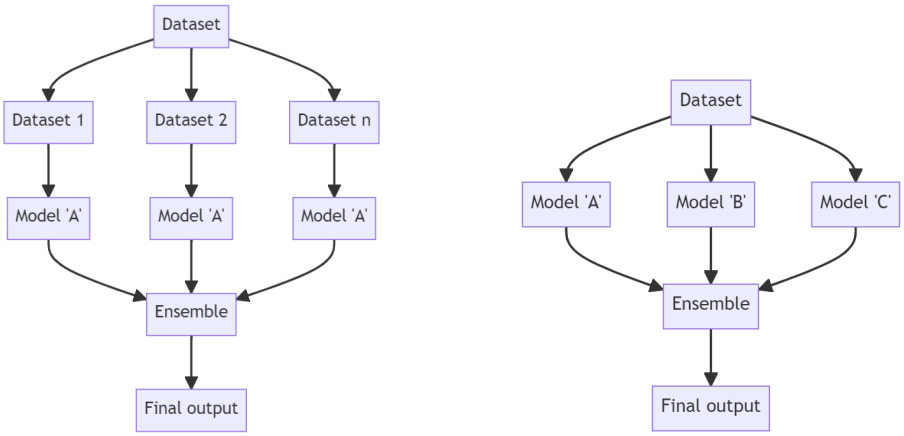

Figure 1.1: Left: Homogeneous EL; **Right:** Heterogeneous EL

Training Methodology

- **Parallel EL**: Weak learners are trained independently of each other. The primary advantages lies in the diversity created through independent training. Moreover, Parallel EL offers parallelization possibilities for the training of the base learners.
- **Sequential EL**: Weak learners are trained sequentially, each one learning from the errors of its predecessors. Therefore, the main advantage of sequential methods is to exploit the dependence between the base learners.

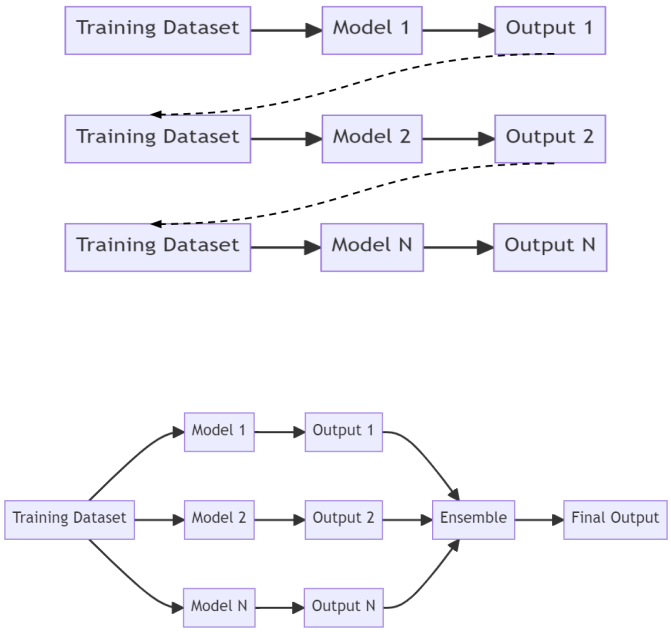

Figure 1.2: Top: Sequential EL; **Bottom:** Parallel EL

1.1.4 Aggregation Mechanisms

Once the set of weak learners is constructed, the next task is to define the aggregation function φ. Several commonly used methods are:

- **Majority Voting**: In the case of classification problems, the label that receives the majority of votes is chosen.
- **Soft Voting**: Probabilities of each label are averaged, and the label with the highest average probability is chosen.

- **Learners Averaging**: The outputs from the weak learners are averaged.

$$\varphi = \frac{1}{N}\sum_{i=1}^{N} h_i \tag{1.2}$$

- **Learners Weighted Averaging**: A weighted average of the outputs is taken, where the weights are learned during a training phase.

$$\varphi = \sum_{i=1}^{N} w_i h_i, \quad \text{where } \sum_{i=1}^{N} w_i = 1 \tag{1.3}$$

1.1.5 Review of Principal Ensembling Methods

Ensemble methods can be primarily categorized into three main types: Bagging, Boosting, and Stacking. Each of these methods has its own unique approach in building an ensemble of weak learners to form a strong learner. They differ substantially in the way they train these weak learners and aggregate their predictions. Below each of these methods is briefly discussed.

Bootstrap Aggregating (Bagging)

Introduced by Leo Breiman in 1996 [3], the Bagging algorithm takes bootstrap samples from the data set and trains a model on each of these samples. In the case of classification problems, a majority vote is taken, while for regression problems, the final prediction is often obtain through learners averaging aggregation.

$$H(x) = \frac{1}{N}\sum_{i=1}^{N} h_i(x) \tag{1.4}$$

Where $H(x)$ is the final prediction and $h_i(x)$ is the i^{th} base learner's prediction.

Random Forests

Random Forests, an application of bagging to tree learners, were proposed by Ho in 1995 [9] and further developed by Breimann (2001) [10]. While traditional bagging uses all features to make the best split in decision trees, Random Forests choose a random subset of features at each split, making the trees more independent and diverse. This is an effective way to control the variance in tree learning, since when dealing with very deep trees they tend to overfit easily the training dataset.

Boosting

Boosting methods train learners in a sequential manner. Each new model corrects the errors of its predecessor, adapting based on the misclassifications or residuals from the previous models. Below, two of the most commonly used boosting algorithms and their characteristics are explored.

AdaBoost

AdaBoost, short for "Adaptive Boosting," was one of the first successful boosting algorithms. It was proposed by Yoav Freund and Robert Schapire [4]. AdaBoost focuses on improving the performance of decision trees on binary classification problems. The algorithm adjusts the weights of misclassified data points at each iteration, encouraging the model to focus on harder-to-classify examples. The final prediction is a weighted sum of the predictions from individual learners.

$$H(x) = \sum_{i=1}^{N} \omega_{it} h_i(x) \tag{1.5}$$

Where $H(x)$ is the final prediction, ω_i is the weight of the i^{th} weak learner, and $h_i(x)$ is the i^{th} weak learner.

Gradient Boosting

Gradient Boosting, proposed by Jerome Friedman [11], generalizes AdaBoost by allowing optimization of arbitrary differentiable loss functions. In essence, each new model fits to the residuals of the combined ensemble of existing models. This allows the method to be used both for regression and classification problems.

$$H_m(x) = h_{m-1}(x) + \sum_{i=1}^{N} \alpha_m \nabla L(y, H_{m-1}(x)) \tag{1.6}$$

Where $L(y, H(x))$ is the loss function, α_m is an optimizable multiplier, and $\nabla L(y, H(x))$ represents its gradient.

Stacking

Stacking, also known as "Stacked Generalization," has been introduced by Wolpert in 1922 [5]. is an ensemble technique in which after having trained several weak learners, a second-level model, also known as "meta-learner", is trained to make a final prediction based on the predictions of the ones obtained from the base learners during the first training phase.

$$H(x) = f(h_1(x), h_2(x), \ldots, h_N(x)) \tag{1.7}$$

Here $H(x)$ is the final prediction, f is the meta-learner, and $h_i(x)$ are the base learners.

Multi-level Stacking

Some stacking implementations take it a step further by implementing multiple layers of meta-learners, each taking as input the predictions of the previous layer and providing a set of predictions for the next layer.

$$H(x) = f_k(\ldots f_2(f_1(h_1(x), h_2(x), \ldots, h_N(x)))\ldots) \tag{1.8}$$

1.1.6 Ensemble Building as a Combinatorial Problem

The process of ensemble building can be viewed through the lens of a combinatorial optimization problem. In essence, the objective is to find the optimal combination of base learners that maximizes the performance of the ensemble while minimizing overfitting and computational cost. This involves selecting from a large set of possible base learners, each with its own hyperparameters, and determining the best way to aggregate their predictions. The space of all possible combinations grows exponentially with the number of base learners and their configurations, making this a non-trivial task.

This combinatorial optimization perspective on ensemble building naturally leads us to the realm of Quadratic Unconstrained Binary Optimization (QUBO) problems. QUBO provides a mathematical framework for solving such complex optimization problems and can be particularly useful for efficiently navigating the combinatorial space of ensemble configurations. The following section will delve deeper into QUBO problems and explore their applicability in optimizing ensemble learning methods.

1.2 QUBO problems

Quadratic Unconstrained Binary Optimization (QUBO) problems serve as a versatile framework for tackling a broad array of combinatorial optimization challenges. In these problems, the objective is to find a vector $\mathbf{v}$ that minimizes a quadratic function. The function is generally represented as:

$$\min_{\{v\}} \left(\sum_i a_i v_i + 2 \sum_i \sum_{j>i} b_{i,j} v_i v_j + c \right) \tag{1.9}$$

Here, v_i are binary variables taking values in $\{0, 1\}$, a_i are the linear coefficients, $b_{i,j}$ are the quadratic coefficients, and c is a constant term. This formulation allows for a wide range of real-world problems to be encoded into a QUBO structure, making it a powerful tool for optimization. For an extensive survey of QUBO problems applications and solutions methods see [12].

The quadratic form $\mathcal{H}$ to be minimized can also be expressed in matrix-vector form as:

$$\mathcal{H}(\mathbf{v}) = \mathbf{v}^T Q \mathbf{v} \tag{1.10}$$

The matrix Q is a symmetric matrix, where the diagonal elements correspond to the linear coefficients a_i and the off-diagonal elements represent the quadratic coefficients $b_{i,j}$:

$$Q = \begin{pmatrix} a_1 & b_{1,2} & b_{1,3} & \cdots & b_{1,N} \\ b_{1,2} & a_2 & b_{2,3} & \cdots & b_{2,N} \\ b_{1,3} & b_{2,3} & a_3 & \cdots & b_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{1,N} & b_{2,N} & b_{3,N} & \cdots & a_N \end{pmatrix} \tag{1.11}$$

1.2.1 Equivalence between QUBO and Ising model

The QUBO framework is closely related to Ising Models:

$$H(\mathbf{s}) = -\sum_{\langle i,j \rangle} J_{ij} s_i s_j - \sum_i h_i s_i \tag{1.12}$$

where s_i are spin variables taking values in $\{-1, 1\}$, J_{ij} are the interaction coefficients, and h_i are the external magnetic fields. The transformation between QUBO and Ising models can be achieved through a simple change of variable:

$$s_i = 2v_i - 1 \tag{1.13}$$

Both QUBO and Ising models are frameworks for solving combinatorial optimization problems, their suitability varies depending on the problem at hand as well as the available solvers. QUBO formulations are often more natural for problems that are inherently binary, such as task scheduling or network design. On the other hand, Ising models are more suitable for problems that can be mapped to physical systems, such as spin glasses or protein folding. QUBO and Ising formulations of many different NP-complete and NP-hard problems are reported in [12] and [13]

1.2.2 Computational Complexity of QUBO Problems

As demonstrated in [14], QUBO problems belong to the complexity class FP^{NP}. This class comprises functional problems that can be solved by a deterministic Turing machine operating in polynomial time, provided it has access to an oracle for NP-complete problems. This means that a solution for such problems can be computed in polynomial time if one has access to a so-called "NP oracle" which is a "magic box" that can instantaneously solve queries related to NP-complete problems.

However, no polynomial-time algorithm is currently known for solving general *NP* problems unless the $P = NP$ conjecture is proven to be true.

Given the inherent complexity of QUBO problems, a variety of algorithms have been developed to tackle them, as documented in [12]. In this thesis work, focus will be on Annealing-based approaches. Specifically, Simulated Annealing (SA) will be considered for the implementation in the context of digital solvers. An overview of Quantum Annealing (QA) instead will be will be given with the prospect of future implementation in the context of quantum, quantum-hybrid, and quantum-simulator solvers.

1.3 Annealing

Annealing is a concept that originates from thermodynamics, particularly in the process of slowly cooling a material to remove defects and optimize the arrangement of its atoms. The term has been metaphorically extended to the field of optimization algorithms to describe methods that seek to find the most optimal solution to a problem by exploring the solution space in a structured yet probabilistic manner.

In the physical process of annealing, a material is heated to a high temperature where its atoms become highly disordered. The material is then slowly cooled, allowing the atoms to settle into a low-energy, highly ordered state. The key to this process is the slow cooling rate, which allows the system to escape local minima in its energy landscape and eventually find the global minimum, corresponding to the most stable atomic configuration.

1.3.1 Simulated Annealing

The concept of annealing has been adapted to solve complex optimization problems in computer science. The most famous algorithm inspired by annealing is simulated annealing, introduced in 1983 [15]. In SA, the objective function to be minimized plays a role analogous to the Hamiltonian in physics. The algorithm starts with a random solution and iteratively moves to neighboring solutions. The acceptance of a new solution is controlled by a parameter analogous to temperature, which decreases over time.

In the original formulation by Kirkpatrick et al., the probability P of accepting a new solution s' from a current solution s is given by:

$$P(s \to s') = \begin{cases} 1 & \text{if } E_{s'} < E_s, \\ e^{-\frac{E_{s'} - E_s}{kT}} & \text{otherwise.} \end{cases}$$

Algorithm 1.1 Simulated Annealing Algorithm

Require: Initial state, initial temperature, cooling rate
Ensure: Optimal state minimizing f

```
Initialize: current_state, temperature, cooling_rate
while temperature > 1
    new_state = neighbor(current_state)
    Δf = f(new_state) − f(current_state)
    if Δf < 0 or exp(−Δf/temperature) > random(0, 1)
        current_state = new_state
    end if
    temperature = temperature * cooling_rate
end while
return current_state
```

1.3.2 Quantum Annealing

Adiabatic Quantum Computing (AQC) is a computational paradigm that heavily relies on the principles of Quantum Annealing (QA). The idea is to encode the solution to a computational problem in the ground state of a quantum Hamiltonian and then evolve the system according to the Adiabatic Theorem (AT).

The adiabatic approximation states that if a quantum system is initially in an eigenstate $|\varepsilon_0(0)\rangle$ of a time-dependent Hamiltonian $H_0(t)$, its evolution, as governed by the Schrödinger equation

$$i\frac{\partial|\psi(t)\rangle}{\partial t} = H(t)|\psi(t)\rangle$$

will approximately keep the system in the corresponding instantaneous ground state $|\varepsilon_0(t)\rangle$ of $H(t)$, provided that $H(t)$ changes "sufficiently slowly."

Quantifying the exact "slowness" of this variations is the subject of AT. It provides a sufficient condition for the success of the computation and, in doing so, provides the run time of a computation in terms of the gaps between eigenvalues Δ of the Hamiltonian and the Hamiltonian's time-derivative. The run time t_f of an adiabatic algorithm scales at worst as $O(1/\Delta^3)$, and if the Hamiltonian is varied sufficiently smoothly, this can be improved to $O(1/\Delta^2)$ up to a polylogarithmic factor in Δ [16].

In [13] can be found an estimate of the "slowness" of the adiabatic evolution dependant on the number of variables N involved in the optimization problem at hand:

$$T = O\left[\exp\left(\alpha N^{\beta}\right)\right] \tag{1.14}$$

in order for the system to remain in the ground state, for positive coefficients α and β, as $N \to \infty$.

Algorithm 1.2 Quantum Annealing Algorithm scheme

Require: Initial Hamiltonian H_{initial}, problem Hamiltonian H_{problem}
Ensure: Optimal quantum state
1: Initialize: quantum system in ground state of H_{initial}, $s = 0$
2: **while** $s < 1$
3: $H(s) = (1 - s)H_{\text{initial}} + sH_{\text{problem}}$
4: Evolve quantum system under $H(s)$ according to the Schrödinger equation
5: $s = s + \Delta s$
6: **end while**
7: **return** Measured quantum state

The question of whether Quantum Annealing (QA) can significantly outperform traditional algorithms like Simulated Annealing (SA) is still a matter of active debate [17]. This issue has attracted considerable research attention, especially with the advent of quantum annealers and their simulators developed by various private companies. The field is marked by divergent findings and continuous advancements in both quantum and classical computing technologies. As such, it remains a dynamic and rapidly evolving area of research.

1.4 QBOOST ALGORITHM

The work presented in this thesis and the algorithms developed during my internship at Axyon AI are deeply influenced by the research conducted by Neven et al. Their paper, titled "Training a Binary Classifier with the Quantum Adiabatic Algorithm" together with following papers serve as foundational guides for trying to implement an improved version of the ensemble building step in the Axyon ML pipeline, [2] [18] [1].

Algorithm 1.3 Qboost ($T \leq Q$) - from Neven et al. [1]

Require: Training and validation data $\{x_s\}$, dictionary of weak classifiers $\{h_i(x)\}$, regularization parameters λ_{min}, λ_{step}, and λ_{max}

Ensure: Strong classifier $H_\omega(\mathbf{x})$

1: Initialize: $\forall s, d_{\text{inner}}(s) = \frac{1}{S}$, $T_{\text{inner}} = 0$; empty strong classifier $H_\omega(\mathbf{x})$; storage for a pool of Q candidate weak learners $\{h_q(x)\}$
2: **repeat**
3: Optimize the members of the dictionary $\{h_i(x)\}$ according to the current $d_{\text{inner}}(s)$
4: From $\{h_i(x)\}$ select the $Q - T_{\text{inner}}$ weak classifiers that have the smallest training error rates
5: **for** $\lambda_{\text{min}} : \lambda_{\text{step}} : \lambda_{\text{max}}$
6: Optimize $\omega^{opt} = arg\min_{\omega} \left(\sum_{s=1}^{S} \left(\frac{1}{N} y_s \sum_{i=1}^{N} \omega_i h_i(\mathbf{x}_s) - 1 \right)^2 + \lambda \|\omega\|_0 \right)$
7: Set $T_{\text{inner}} = \|\omega\|_0$
8: Construct strong classifier $H(x) = sign\left(\sum_{i=1}^{N} \omega_i h_i(x) \right)$
9: Measure validation error of $H(x)$ on unweighted validation set
10: **end for**
11: Save ω^{opt}, T_{inner}, $H(\mathbf{x})$ and the validation error from the optimization run that achieved the lowest validation error so far
12: Update $d_{\text{inner}}(s) = d_{\text{inner}}(s) \left(y_s \sum_{i=1}^{N} \omega_i h_i(\mathbf{x}_s) - 1 \right)^2$
13: Normalize $d_{\text{inner}}(s)$
14: Delete from the pool $\{h_q(x)\}$ the $Q - T_{\text{inner}}$ weak learners for which $\omega = 0$
15: **until** Validation error stops decreasing

In particular, in the original paper by Neven [2] the strong classifier prediction is given by:

$$y = H(x) = sign\left(\sum_{i=1}^{N} \omega_i h_i(x)\right) \tag{1.15}$$

where $x \in \mathbb{R}^M$ are the input patterns to be classified, $y \in \{-1, 1\}$ is the output of the classifier, $h_i : x \mapsto \{-1, 1\}$ are the weak classifiers and $\omega_i \in \{0, 1\}$ are the set of binary weights to be optimized during QUBO solving procedure.

It is clear by looking at Alg.1.3, that the problem of building the ensemble is reduced to iteratively solve the following QUBO problem:

$$\omega^{opt} = \underset{\omega}{arg\min}\left(\sum_{s=1}^{S}\left(\frac{1}{N} y_s \sum_{i=1}^{N} \omega_i h_i(\mathbf{x}_s) - 1\right)^2 + \lambda\|\omega\|_0\right) \tag{1.16}$$

The term S refers to the number of samples, and λ is a regularization parameter that controls the numerosity of the ensemble penalizing the L_0 norm of the weights array ω . The objective function 1.16 aims to minimize the squared error between the binary label y_s and the average of the binary predictions produced by the weak learners. The re-weighting follows the conventional boosting procedure described in 1.1.5, with a significant distinction: the process operates over a pool of base learners, rather than optimizing each learner sequentially.

The training dataset is dynamically re-weighted according to the predictive performance of the current strong classifier $H(x)$ over each data sample x_s.

The experiments conducted in [2] and in [1] shows that Qboost often outperforms AdaBoost solidly, especially in terms of classifier compactness, while maintaining comparable accuracy.

2

Problem framework

Axyon AI is a technology company specialized in the development of artificial intelligence models for quantitative asset management. The company leverages advanced machine learning techniques to optimize financial portfolios, manage risks, and build investment strategies. One of the key ML framework employed by Axyon AI is Learning to Rank (LtR), which is instrumental in sorting and prioritizing assets based on multiple features and criteria.

	t		t+H	
Asset	Model Score	Axyon Rank	H day Return	Realized Rank
A	0.457	6	-3%	6
B	0.407	7	-2%	5
C	0.529	4	-6%	8
D	0.494	5	2%	2
E	0.622	1	5%	1
F	0.581	2	-5%	7
G	0.541	3	1%	3
H	0.367	8	-1%	4

Figure 2.1: Example of LtR data sample exploited at Axyon AI. Here H is the time horizon at which the asset returns are evaluated.

2.1 Learning to Rank Methods

Learning to Rank (LtR) is a specialized paradigm within ML tailored for ranking. It is fundamentally concerned with constructing a model capable of arranging a set of items in a specific order based on certain criteria. The LtR methods can be broadly classified into three categories: Point-wise, Pairwise, and List-wise methods.

1. **Point-wise Methods**: Point-wise approaches treat the ranking task as either a regression or classification problem. Each item is independently scored, and these scores are subsequently utilized for sorting the items. The mathematical formulation can be represented as $f(x) \rightarrow y$, where f is the ranking function, x is the feature vector, and y is the score.
2. **Pairwise Methods**: Pairwise approaches consider the relative ordering of item pairs. The objective is to minimize the number of incorrectly ordered item pairs. RankNet [19] is a commonly employed algorithm in this category, which has the merit to exploit a differentiable loss function.
3. **List-wise Methods**: List-wise methods take into account the entire list of items and aim to optimize a list-wise loss function. Normalized Discounted Cumulative Gain (NDCG) is often used as the evaluation metric. Algorithms like ListNet [20] are popular choices for List-wise ranking.

Axyon AI employs a combination of these methods to build robust ranking models for asset management.

2.2 Multilabel Classification

Axyon AI has defined an effective way to map LtR problems to multi-label classification problems.

In ML, classification tasks are categorized into different types such as binary, multi-class, and multi-label classification. While binary and multi-class classification aim to classify instances into one of two or more classes, multi-label classification is concerned with assigning a set of target labels to each instance. In other words, an instance can belong to multiple classes simultaneously.

Formal Definition

Mathematically, a multi-label classification problem is defined as follows:

Let X be the feature space such that $X \subseteq \mathbb{R}^d$, where d is the number of features. Let Y be the label space where $Y = \{y_1, y_2, \ldots, y_k\}$, with k representing the number of distinct labels.

A training set $D = \{(x_1, Y_1), (x_2, Y_2), \ldots, (x_n, Y_n)\}$, where $x_i \in X$ is a feature vector, and $Y_i \subseteq Y$ is the set of labels corresponding to x_i.

The goal is to learn a function $h : X \rightarrow 2^Y$, where 2^Y is the power set of Y, such that $h(x_i)$ approximates Y_i as closely as possible for all $x_i \in X$.

Common loss functions for multi-label classification include:

- Hamming Loss:

$$\text{HammingLoss}(Y, h(x)) = \frac{1}{|Y|} \sum_{y \in Y} I(y \in Y \oplus y \in h(x))$$

- Jaccard Loss:

$$\text{JaccardLoss}(Y, h(x)) = 1 - \frac{|Y \cap h(x)|}{|Y \cup h(x)|}$$

Converting Multi-label to Binary Classifications

One common approach to solve multi-label classification problems is to decompose them into multiple binary classification problems. For each label $y_i \in Y$, a separate binary classifier $h_i : X \to \{0, 1\}$ is trained. The final multilabel classifier $h(x)$ aggregates the output of all binary classifiers as $h(x) = \{y_i | h_i(x) = 1, i = 1, \ldots, k\}$.

2.3 Ensemble Learning with Greedy Algorithms

Greedy algorithms form a category of algorithms that make locally optimal choices at each step, aiming for a global optimum. These algorithms are efficient and easy to implement but are not always guaranteed to find the optimal solution. In fact, the portion of the space of possible configurations explored with this kind of procedure is tiny with respect to the size of the space itself, and the risk of reaching a sub-optimal solution is considerably high. In particular, we can make a rough estimate of the fraction of space explored by considering the total number of possible configurations for an ensemble of K models formed starting from a pool of N candidates and the number of configurations actually screened through this procedure:

$$\text{portion of conf. space explored} = \frac{N^K}{2^N}$$

In the context of Axyon AI, a greedy iterative ensemble selection approach tailored for multi-label classification problems is exploited. This procedure is inspired by the works [21] & [22], and aims to optimize ensemble performance iteratively.

Greedy Iterative Ensemble Selection Procedure

Typically, the procedure involves:

- $N = 5000$ candidate models.
- $T = 24$ meta-validation runs, each lasting 3 months.
- Scores $s_t^{(i)}$ represent a specific performance metric of model i in run t.
- f is generally a risk-adjusted performance metric computed as $S^{(\cdot)} = \frac{\overline{(s_t^{(\cdot)} - s_{\text{baseline}})}}{\sigma(s_t^{(\cdot)})}$, where s_{baseline} is a baseline value.
- Filtering is done based on a one-tailed t-test, comparing each model against a random baseline.

The algorithm proceeds as outlined in Alg.2.1.

Algorithm 2.1 Greedy Iterative Ensemble Selection Procedure

Require: Pool of N candidate models with scores $s_t^{(i)}$ across T time periods, statistical function f, maximum ensemble size K
Ensure: Optimized ensemble $E_{[k^*]}$
1: Initialize: Empty ensemble $E_{[0]}$, pool of N candidates, set $k^* = 0$
2: Compute summary scores $S^{(i)} = f(s_1^{(i)}, s_2^{(i)}, \ldots, s_T^{(i)})$ for each candidate model i
3: Filter out candidates based on threshold or statistical test on $S^{(i)}$
4: Find top-performing model i^* with highest $S^{(i^*)}$
5: Set $E_{[1]} = i^*$, $k^* = 1$
6: Remove i^* from the pool of candidate models
7: **repeat**
8: **for** each remaining candidate j
9: Form potential ensemble $E_{[k^*+1]}$ by combining $E_{[k^*]}$ and j
10: Compute new ensemble score $S^{(E_{[k^*+1]})}$ by evaluating the combined model performance
11: If $S^{(E_{[k^*+1]})} > S^{(E_{[k^*]})}$, then set $E_{[k^*+1]} = E_{[k^*]} \cup j$, $k^* = k^* + 1$
12: **end for**
13: Remove models added to $E_{[k^*]}$ from the pool of candidate models
14: **until** $k^* = K$ or no more candidates remain

Evaluation metrics

Examples of metrics widely used in finance to evaluate quantitative investment strategies are Sortino and Sharpe ratios. In Axyon AI context Rank-ic metric is exploited to evaluate the performance of candidate models and ensembles, together with risk-adjusted versions of it that resemble in their definitions Sortino and Sharpe ratios.

In particular:

- **Rank-IC (RIC):** The Rank-IC (RIC) is a specialized application of Spearman's Rank Correlation Coefficient, designed to measure the linear correlation between two sets of ranked data. It evaluates the monotonic relationship between the two variables, quantifying both the strength and the direction of this relationship. The RIC is mathematically defined as:

$$\text{Rank-IC}_t = \rho_{\text{R}(X)_t, \text{R}(Y)_t} = \frac{\text{cov}(\text{R}(X)_t, \text{R}(Y)_t)}{\sigma_{\text{R}(X)_t} \sigma_{\text{R}(Y)_t}} \tag{2.1}$$

  where $\text{R}(X_i)$ and $\text{R}(Y_i)$ denote the ranks of the variables X and Y, respectively. The coefficient can take values from -1, representing a perfect negative correlation, to +1, indicating a perfect positive correlation.

- **Sortino Ratio:** The Sortino Ratio serves as an indicator of the risk-adjusted performance of an investment asset or portfolio. It contrasts the asset's return against the downside risk. The ratio is defined as follows:

$$\text{Sortino Ratio} = \frac{R - T}{DR} \tag{2.2}$$

  Here, R is the asset's return, T is the target or required rate of return, and DR is the downside standard

deviation. In the Axyon framework instead of the Asset's return it is considered the Rank-IC associated with the model predictions and its downside standard deviation.

- **Sharpe Ratio:** The Sharpe Ratio is another metric that evaluates risk-adjusted returns. Unlike the Sortino Ratio, it considers the total volatility of the investment. The ratio is calculated as:

$$\text{Sharpe Ratio} = \frac{R - R_f}{\sigma} \tag{2.3}$$

In this equation, R is the asset's return, R_f is the risk-free rate, and σ represents the asset's standard deviation.

In the Axyon framework instead of the Asset's return it is considered the Rank-IC associated with the model predictions and its downside standard deviation.

2.3.1 Creating a Benchmark

The greedy ensemble selection procedure has been refactored in order to optimize LtR metrics obtained directly from the ensembled predictions and not, as it is originally done, optimize aggregated metrics over long period of time for each of the weak learner considered in the procedure.

This adaptation aims to establish a reliable benchmark against which we can compare the Qboost implementation.

This more direct approach ensures a more aligned comparison when evaluating the performance of Qboost, which considers directly the multi-label predictions of the learners.

3

Qboost for Multi-Label classification and Learning to Rank

Given the description of the original Qboost algorithm given in section 1.4 here we want to describe how it has been modified in order to be exploited in the context of multi-label classification and LtR problems.

3.1 Qboost on multi-label classification problem

In order to frame the ensembling of weak classifiers built for the multi-label classification problem into the Qboost optimization procedure we note that the former can be interpreted as a sequence of B binary classifiers that work simultaneously on the same data sample $\mathbf{x}_s$. In light of this the first try we perform is to modify the loss in Eq.1.16 in the following way:

$$\omega^{opt} = \underset{\omega}{arg\min} \left(\sum_{s=1}^{S} \sum_{k=1}^{B} \left(\frac{1}{N} y_{s,k} \sum_{i=1}^{N} \omega_i h_{i,k}(\mathbf{x}_s) - 1 \right)^2 + \lambda \|\omega\|_0 \right) \tag{3.1}$$

Expanding the square, re-arranging the sums and exploiting the property for which $\omega^2 = \omega$, we obtain that the function $\mathcal{H}$ that has to be minimized can be written as the following quadratic form:

$$\begin{aligned} \mathcal{H} &= \sum_{i=1}^{N} \sum_{j>i}^{N} \omega_i \omega_j \left[\sum_{s=1}^{S} \sum_{k=1}^{B-1} h_{i,k}(\mathbf{x}_s) h_{j,k}(\mathbf{x}_s) \right] + \sum_{i=1}^{N} \omega_i \left[\frac{SK}{N^2} - \frac{2}{N} \left(\sum_{s=1}^{S} \sum_{k=1}^{B-1} y_{s,k} h_{i,k}(x_s) \right) + \lambda \right] + SK \\ &= \omega^T Q \omega \end{aligned} \tag{3.2}$$

where Q is our QUBO matrix.

Moreover, modifications have been made to the boosting step of Algorithm 1.3 by introducing a concept termed 'weak boosting.' On the one hand, this avoids the retraining of the pool of weak learners $\{h_i\}$ at each iterative step, a procedure that is surelly more effective but infeasible within the Axyon framework. On the other hand, it aims to still exploit all the heterogeneity present in the pool of weak learners. Specifically, the procedure commences with a set of pre-trained learners. At each optimization cycle, the importance of the weak learners' predictions over a particular data sample is reweighted based on the performance of the strong ensemble on that particular sample. This approach is expected to significantly impact the optimization under specific conditions: either if the size of the pool of weak learners $\{h_i\}$ considerably exceeds that of the set of weak learners considered for optimization $\{h_q\}$, or if $\{h_i\}$ contains members with a high level of heterogeneity.

The entire optimization procedure is outlined in Algorithm3.1

Algorithm 3.1 Qboost multi-label- inner cycle on Axyon data

Require: Validation and test data $\{x_s\}$ related to the performance of a dictionary of weak classifiers $\{h_i(x)\}$, regularization parameters λ_{min}, λ_{step}, and λ_{max}

Ensure: Strong classifier $H_\omega(\mathbf{x})$

Compute multi-label- representation of the predictions of $\{h_i(x)\}$ over $\{x_s\}$

Compute multi-label- representation of the real assets performances contained in $\{x_s\}$

Compute accuracy values $P(h_i(x_s))$ for each sample in $\{x_s\}$

Initialize: $\forall s, d_{\text{inner}}(s) = 1$, $T_{\text{inner}} = 0$; empty strong classifier $H_\omega(\mathbf{x})$

repeat

  Compute $P(x_s) \cdot d_{\text{inner}}(s) \forall x_s \in$ validation data

  From $\{h_i(x)\}$ select the $Q - T_{\text{inner}}$ weak classifiers that have the highest weighted accuracy over validation data.

  for $\lambda_{\text{min}} : \lambda_{\text{step}} : \lambda_{\text{max}}$

    Optimize $\omega^{opt} = \underset{\omega}{arg\min} \left(\sum_{s=1}^{S} \sum_{k=1}^{B-1} \left(\frac{1}{N} y_{s,k} \sum_{i=1}^{N} \omega_i h_{i,k}(\mathbf{x}_s) - 1 \right)^2 + \lambda \|\omega\|_0 \right)$

    Set $T_{\text{inner}} = \|\omega\|_0$

    Construct strong classifier $H(x)$ by computing the multi-label- representation of $\sum_{i=1}^{N} \omega_i h_i(\mathbf{x}_s)$

    Measure accuracy of $H(x)$ on unweighted test set

  end for

  Save ω^{opt}, T_{inner}, $H(\mathbf{x})$ and the validation error from the optimization run that achieved the highest test accuracy so far

  Update $d_{\text{inner}}(s) = 1 - P(H(x_s))$

  Delete from the pool $\{h_q(x)\}$ the $Q - T_{\text{inner}}$ weak learners for which $\omega = 0$

until Validation error stops decreasing

3.2 QBOOST ON A LEARNING TO RANK PROBLEM

As outlined in Chapter 2 the framework under which we are trying to solve the ES problem is composed of two well distinct step:

- **The multi-label classification problem**, that the learners directly solves. Meaning to say that the learners predictions reguards this problem.
- **The LtR problem**, that is solved through a process that build the rankings from the predictions of the learners.

The QUBO problem described by eq.3.1 clearly refers to the first problem step, trying to minimize the error made by the ensemble in the multiple binary classifications it performs over the data. However, the truly final result of the ML process is the ranking, and it is the accuracy of the ranking that defines how successful our EL process is. This consideration has led to the attempt to formulate a QUBO problem that takes into account directly the rankings.

In order to do this we tried to include the covariance between the predicted and realized rankings in the quadratic form that defines the QUBO problem.

In particular, the covariance squared between two finite-dimensional arrays that defines the rankings seems to naturally define a QUBO problem:

$$\mathcal{H}_{Cov} = -\sum_{s=1}^{S} \mathrm{Cov}^2\left(\vec{x_s}, \vec{y_s}\right) + \lambda \sum_{i=1}^{N} \omega_i = -\sum_{s=1}^{S} \mathrm{Cov}^2\left(\frac{1}{N}\sum_{i=1}^{N} \omega_i \vec{h_{i,s}}, \vec{y_s}\right) + \lambda \sum_{i=1}^{N} \omega_i \tag{3.3}$$

where the covariance is defined as:

$$\mathrm{cov}(X, Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - E(\mathbf{X}))(y_i - E(\mathbf{Y}))$$

(3.4)

and the expected value $E(\mathbf{X})$ is given by the average value since we take a uniform probability distribution.

Some specifications must be made. Covariance is not a limited quantity, it can span in the range $[-\infty; \infty]$ and therefore a selection must be made before the QUBO matrix is built. In fact, we are interested only in the learners, hence in the ensembles, that express a positive correlation with the realized rankings. Therefore, we have to eliminate from the pool of available weak learners all the ones that shows a negative average covariance with the realized rankings.

If we proceed to expand Eq. 3.3 through the Eq:3.4, we find:

$$\mathcal{H}_Q^{Cov} = -\sum_{s=1}^{S} \frac{1}{P_s^2} \left[\sum_{k=1}^{P_s} \sum_{z=1}^{P_s} \left(x_{s,k}x_{s,z}y_{s,k}y_{s,z} - x_{s,k}x_{s,z}y_{s,k}\bar{y}_s - x_{s,k}y_{s,k}y_{s,z}\bar{x}_s + x_{s,k}y_{s,k}\bar{x}_s\bar{y}_s - x_{s,k}x_{s,z}y_{s,z}\bar{y}_s + \right.\right.$$
$$+ x_{s,k}x_{s,z}\bar{y}_s^2 + x_{s,k}y_{s,z}\bar{x}_s\bar{y}_s - x_{s,k}\bar{x}_s\bar{y}_s^2 - x_{s,z}y_{s,k}y_{s,z}\bar{x}_s + x_{s,z}y_{s,k}\bar{x}_s\bar{y}_s + y_{s,k}y_{s,z}\bar{x}_s^2 -$$
$$\left.\left. - y_{s,k}\bar{x}_s^2\bar{y}_s + x_{s,z}y_{s,z}\bar{x}_s\bar{y}_s - x_{s,z}\bar{x}_s\bar{y}_s^2 - y_{s,z}\bar{x}_s^2\bar{y}_s + \bar{x}_s^2\bar{y}_s^2 \right) \right] + \lambda \sum_{i=1}^{N} \omega_i \tag{3.5}$$

where P_s represents the number of instruments that must be ranked the day s, that is the dimensionality of the arrays we are working with. This number is slightly variable in most of the investable universe taken into account in real world scenarios.

Below we report the extended form of the building blocks of Eq.3.3:

- $x_{s,k} = \frac{1}{N} \sum_{i=1}^{N} \omega_i h_{i,s,k}$
- $x_{s,k}^2 = \frac{1}{N^2} \left[\sum_{i=1}^{N} \omega_i h_{i,s,k}^2 + \sum_i \sum_{j \neq i} \omega_i \omega_j h_{i,s,k} h_{j,s,k} \right]$
- $\bar{x}_s = \frac{1}{P_s} \frac{1}{N} \sum_{v=1}^{P_s} \sum_{i=1}^{N} \omega_i h_{i,s,v}$
- $\bar{x}_s^2 = \frac{(P_s+1)^2}{4N^2P_s^2} \left[\sum_{i=1}^{N} \omega_i + \sum_{i=1}^{N} \sum_{j=1}^{N} \omega_i \omega_j \right]$
- $x_{s,k}\bar{x}_s = \frac{(P_s+1)}{2N^2P_s} \left[\sum_{i=1}^{N} \omega_i h_{i,s,k} + \sum_{i=1}^{N} \sum_{j=1}^{N} \omega_i \omega_j h_{i,s,k} \right]$
- $x_{s,k}x_{s,z} = \frac{1}{N^2} \left[\sum_{i=1}^{N} \omega_i h_{i,s,k} h_{i,s,z} + \sum_{i=1}^{N} \sum_{j=1}^{N} \omega_i \omega_j h_{i,s,k} h_{j,s,z} \right]$
- $\bar{y}_s = \frac{P_s+1}{2P_s}$
- $\bar{y}_s^2 = \frac{(P_s+1)^2}{4P_s^2}$

With these expressions we are able to write the matrix element of the QUBO that describes Eq:3.3. The full expressions are reported in Appendix.A

From a practical point of view first it has been implemented the QUBO matrix representative of each day, and then all the matrices are summed together. This is done to handle properly the fact that P_s is variable.

After the first implementation of this QUBO problem we tested it on the C10 data-set 4.2. Here we report some considerations over this preliminary test:

- Matrix Q is symmetric.
- Elements Q_{ij} are negative $\forall \{i, j\}$.
- Of course, we can make diagonal elements Q_{ii} positive by tuning λ, in order to make the form of the QUBO matrix non-trivial.
- An exhaustive search was conducted to identify values of λ that would lead to non-trivial solutions $\omega_i = 1 \forall i \in [1; N]$ and $\omega_i = 0 \forall i \in [1; N]$. Despite this effort, the algorithm continued to produce trivial solutions within a narrow range of λ values, specifically between 1.17529925 and 1.17529927.

Brute force QUBO optimization has been employed to investigate the behavior of the system for low-dimensional configuration spaces. The analysis reveals the following patterns:

- When considering the solution where all ω_i are set to 1 (for all i in the range [1; N]), it results in a negative value for the function $\mathcal{H}^{Cov}$. Conversely, if all ω_i are set to 0, $\mathcal{H}_Q^{Cov}$ equals 0. Solutions with intermediate values of ω_i fall between these two extremes.
- As the parameter λ is increased, the non-trivial solutions progressively become positive in an ordered manner. Eventually, the solution where all ω_i are 1 also becomes positive. At this juncture, the optimization algorithm shifts to selecting the alternate trivial solution where all ω_i are 0.

Due to these observations, coupled with the computational complexity of the implementation, we opted not to pursue further extensive testing of this QUBO formulation.

4

Experiments

The primary objective of this chapter is to present a set of experiments that scrutinize and evaluate the efficacy of Multi-label Qboost and QUBO optimization methods in the context of the ES step performed at Axyon. Therefore, There will be a benchmark against Axyon's current operational standard, a greedy algorithm designed for ensemble construction. This benchmarking mechanism serves as both a foundational basis and a comparative backdrop for our experimental evaluation, allowing us to gauge the potential advantages or limitations of implementing Qboost or QUBO algorithms in Axyon's existing technological stack.

4.1 QUBO solver

In optimizing the QUBO matrices, we used D-Wave's Neal library, which implements the simulated annealing algorithm specifically tailored for solving optimization problems in this form. A key feature of the simulated annealing approach is that it allows for the tuning of various hyper-parameters to guide the search in the solution space. Here, we pay close attention to two critical hyper-parameters: the 'number of reads' and the 'number of sweeps.'

The 'number of reads' serves as the number of independent repetitions for the annealing process. Each read is essentially an individual run that could potentially yield a different solution due to its stochastic nature. For our experiments, we used a typical value of 1000 reads, optimizing for a balance between computational burden and the quality of the solution.

The 'number of sweeps' represents the number of Markov Chain Monte Carlo (MCMC) updates attempted in each read. These sweeps are essential for exploring the state space efficiently, especially when dealing with complex multi-dimensional problems like those in ensemble learning. We set this parameter to 2000 sweeps, which we found to be a robust choice through preliminary tests.

The annealing temperature followed a geometric schedule. The initial states for the annealing process were

selected at random, providing a diverse starting point for each read. As for the temperature range, which significantly influences the annealing schedule, its range was determined internally by the solver for each QUBO problem, contingent on the total node bias in the QUBO matrix.

For the empirical studies, we utilized two distinct datasets. The MNIST dataset functioned as a 'toy dataset,' allowing us to understand the functioning of the Qboost algorithm in a more controlled setting. This experience laid the groundwork for the real-world application of the algorithm within Axyon's framework, which involved testing on Axyon's proprietary datasets. These datasets offer complex, real-world challenges and are highly representative of the kinds of problems Axyon aims to solve through ensemble learning.

4.2 Datasets

This study utilizes a diverse range of datasets, including proprietary feeds such as C10 and FTSEMIB, as well as the public MNIST dataset. Each dataset has distinct characteristics that make it suitable for specific aspects of the research.

4.2.1 C10 Dataset

The C10 dataset is a specialized investable feed used at Axyon. This dataset is unique in that it has only 10 constituents for each trading day. It covers a time span from May 2, 2007, to November 26, 2013. The weak learners predict over a 20-day horizon. Out of the 6.4×10^6 total samples, 1.6×10^6 are allocated for testing.

4.2.2 FTSEMIB Dataset

The FTSEMIB dataset is associated with the FTSE MIB index, the leading stock market index for the Borsa Italiana. On average, this dataset comprises 40 constituents for each trading day. There are two versions of this dataset:

- The first version spans from June 4, 2013, to December 23, 2019. It has 1.7×10^7 total samples, with 5.9×10^6 designated for testing. The dataset includes 100 weak learners and features a 20-day prediction horizon.
- The second version covers from June 4, 2013, to December 29, 2016. It comprises 4.4×10^7 total samples, with 1.5×10^7 allocated for testing. This version includes 500 weak learners.

4.2.3 MNIST Dataset

The MNIST (Modified National Institute of Standards and Technology) dataset acts as the toy dataset for our experimentation. Comprising 70,000 samples—60,000 for training and 10,000 for testing—it is primarily used to validate the Qboost algorithm before deploying it into Axyon's proprietary frameworks.

Dataset	Prediction Horizon	No. of Weak Learners	Total Samples	Testing Samples
C10	20 days	100	6.4×10^6	1.6×10^6
FTSEMIB (v1)	20 days	100	1.7×10^7	5.9×10^6
FTSEMIB (v2)	20 days	500	4.4×10^7	1.5×10^7
MNIST	N/A	150	70,000	10,000

Table 4.1: Data-set summary

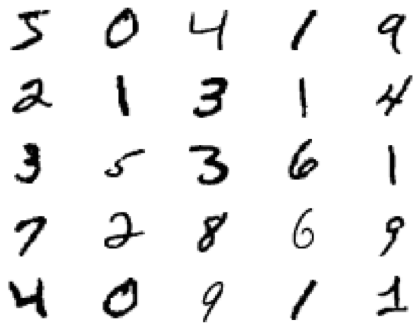

Figure 4.1: Data samples from the MNIST data-set

4.3 Multi-label Qboost implementation

The objective of this section is to provide a comprehensive examination of the Qboost methodology in a multi-label classification context. Specifically, the focus is on the performance of Qboost in generating an effective ensemble of weak learners, which will then be compared against two separate benchmarks: the customized greedy algorithm acting on the model's predictions and, on when considering C10 data-set, the standard greedy algorithm which utilizes aggregated metrics. The latter is the algorithm currently deployed in Axyon's production environment.

4.3.1 Experiments on toy-scale dataset

To evaluate the efficacy of Qboost in a controlled setting, we employed the widely-used MNIST data-set. The main objective was to construct an ensemble that optimizes with respect to the multi-label loss function, as specified in equation 3.1. The Qboost algorithm, particularly its inner cycle, has been implemented as outlined in Algorithm 1.3, with a major difference that involves the update of the weight factor $d_{\text{inner}}(s)$ applied to the training data-set. Instead of using the update procedure in Alg.1.3, it has been exploited the approach proposed by Leclerc et al. [23]. The update equation is expressed as follows:

$$
\begin{aligned}
d_{i+1}(s) &= \frac{1}{Z_i} d_i(s) e^{-\omega_i y_s H_i(x_s)} \\
\omega_i &= \frac{1}{2} \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right) \\
\varepsilon_i &= \sum_s d_i(s) \cdot H_i(x_s) \quad ; \quad H_i(x_s) \neq y_s
\end{aligned}
\tag{4.1}
$$

Here, Z_i serves as the normalization factor ensuring that d_{i+1} is a probability distribution. The index i denotes the current optimization cycle.

Execution and Expectations

In this context, it is important to note that the experiment fully executes the Qboost process, including the boosting step. During this stage, we utilized a series of Keras neural networks—specified in Listing 4.1, which were retrained on the re-weighted data-set. This choice of architecture, featuring a single hidden layer with 32 units, is intentionally simplistic. The aim is to have "weak learners" to validate the effectiveness of Qboost on a relatively straightforward learning task.

The primary expectation from this experiment is a continuous improvement in the ensemble's performance across multiple Qboost optimization cycles. Additionally, it is expected that the average performance of the individual weak learners will remain stable throughout the optimization process.

```
#Network architecture
classifier = Sequential()
classifier.add(tf.keras.layers.Flatten(input_shape=self.input_shape))
classifier.add(tf.keras.layers.Dense(units=32, activation='relu'))
classifier.add(tf.keras.layers.Dropout(rate=0.2))
classifier.add(tf.keras.layers.Dense(units=3, activation='sigmoid'))

#Compile the model with custom accuracy
classifier.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.003),
    loss='binary_crossentropy',
    metrics=[custom_accuracy])
```

Listing 4.1: weak learner

4.3.2 Axyon data

The implementation of the Qboost procedure on Axyon framework differs significantly from the one exploited over MNIST dataset. In fact, it requires the modifications specified in Section 3.1 and follows closely Algorithm 3.1.

Axyon data differs significantly from the MNIST dataset, both in nature and in the computational demands they require to be processed. In particular, it is worth to remind that in this case a proper boosting step is not

performed. Unlike MNIST, where the boosting step was performed by retraining neural networks, in the Axyon framework the accuracy measure of each weak learner over the training samples is re-weighted depending on how the ensemble performs on that particular data sample.

This methodological divergence is a direct consequence of computational constraints and the necessity for greater reproducibility:

- **Computational Time:** To understand the computational burden, consider that for the FTSEMIB data-set spanning from 2013 to 2019, each re-training cycle would take approximately 250 hours for 1000 individuals. Given this, running the boosting process for the whole data-set would be computationally impractical.
- **Reproducibility:** The Axyon environment relies on a set of complex procedures for signal generation. Implementing a boosting step that retrains learners would introduce an extra layer of complexity, making it more challenging to ensure consistent and reproducible results.

The above considerations guided the decision to forgo the boosting step in the implementation of the Qboost procedure for the Axyon framework. Even so, by weighting the predictions of existing weak learners we tried to exploit their diversity at best, in order to build an ensemble as heterogeneous as possible.

Metrics from ensembles generated through the Qboost procedure will be contrasted with those from ensembles constructed via greedy algorithms. This comparison will encompass both the ensemble-building phase and an Out-of-Sample period not previously used for ensemble generation by any algorithm involved in the experiment.

The experiment aims to measure performance disparities between Qboost and greedy algorithms, quantify the effect of "weak boosting" during the construction phase, and investigate the extent of the relationship between accuracy metrics used during ensemble building and those employed for evaluating ensemble performance in the LtR problem.

4.4 QUBO optimizations with a fixed size QUBO matrix

Given the observations that the "weak boosting" mechanisms in the multi-label Qboost algorithm appear to have a limited influence during the ensemble-building stage, as evidenced by the experiment described in Section 4.3.2, it became important to gain a deeper understanding of a single Qboost optimization cycle. Particularly, the outer loop of the Qboost algorithm, detailed in Algorithm 3.1, converges quickly—often within the first or second iteration. This rapid convergence prompted further investigation.

Objectives

The overarching aims of this focused study can be divided into two main categories:

1. **Effects of Varying the Regularization Parameter λ:** The first objective was to explore how adjustments to the regularization parameter λ influence the resulting ensemble's quality. A dense schedule of λ values was adopted to thoroughly cover a wide spectrum of ensemble numerosities.

2. **Stability of Simulated Annealing Solutions:** The second objective aimed to evaluate the robustness of the ensembles generated by the simulated annealing algorithm. Specifically, it examined whether different λ values, which yield ensembles of the same numerosity, lead to the incorporation of different weak learners.

Implications for QUBO Optimization

The quest for such granularity in λ and its impact is driven by the utility of understanding how sensitive the QUBO optimization process is to changes in the regularization parameter. If it turns out that various λ values produce ensembles of identical numerosity but with different weak learners, it would suggest that the solution space explored by the simulated annealing algorithm is populated by many similar local minima generated by different weak learners. This could, in turn, suggest that the pool of available weak learners may need diversification for more robust optimization.

4.5 Configuration space of different dimensions

This experiment is performed in order to verify both the optimization capability of the simulated annealing optimizer and the sensibility of the performance with respect to the dimensions of the configuration space that we consider for the optimization.

Methodology

In order to do this we perform a single optimization cycle of the Qboost algorithm 3.1, varying the regularization parameter λ, for QUBO matrices of different sizes. For each QUBO matrix size the best performing ensemble out of the ones obtained varying λ is chosen.

We compare the results with the ones obtained through the greedy algorithm performed over the predictions, asking for an ensemble with maximum size equal to the size of the QUBO matrix we optimize.

For low dimensional configuration space we produce also the brute force solution of the QUBO problem in order to test whether the Qboost optimization cycle is able to produce the same solution and if it is the one chosen by the algorithm. Moreover, we report the brute force solution of the combinatorial problem for low dimensional configuration spaces obtained for a null value of the regularization parameter. This is done in order to have a catch of what would be the solution without constraining the numerosity of the ensemble.

Expectations

We expect that the scores with which we measure the performance of the ensemble can only increase in the case of the greedy algorithm, while for the QUBO optimization procedure it can also decrease because the objective function we are trying to minimize does not focus directly on these metrics, that regards the rankings builded with the predictions of our ensembles, instead it focus on the accuracy of the ensemble in the multi-label classification problem that the ensemble directly solves.

5 Results

In this chapter, we present the outcomes of the investigative work carried out during this study, specifically focusing on the experiments elaborated in Chapter 4.

5.1 Multilabel Qboost implementation

In this section we report the results obtained from the experiment described in section 4.3. In particular, we will show the results obtained running our multi-label Qboost implementation over the MNIST dataset and over the Axyon data. In this last case, benchmarking is conducted in two ways: against the greedy algorithm described in Section 2.3 during the ensemble-building process, and also among the resulting ensembles themselves.

5.1.1 Toy dataset - MNIST

we report the results of a Qboost run with the following parameters:

- Each learner is trained over 10 epochs with a batch size of 4000 samples.
- $h_i(x)$: the pool of weak learner from which we sample the one to be optimize at each **repeat** cycle is made of 150 learners.
- $h_q(x)$: the number of learners considered at each optimization cycle is 50.
- T: The desired final ensemble size is of 20 learners.
- λ: We consider three values of the regularization parameter at each **repeat** cycle, [2100,2300,2500].
- The annealing parameters are 1000 reads and 2000 sweeps with a geometric β schedule.

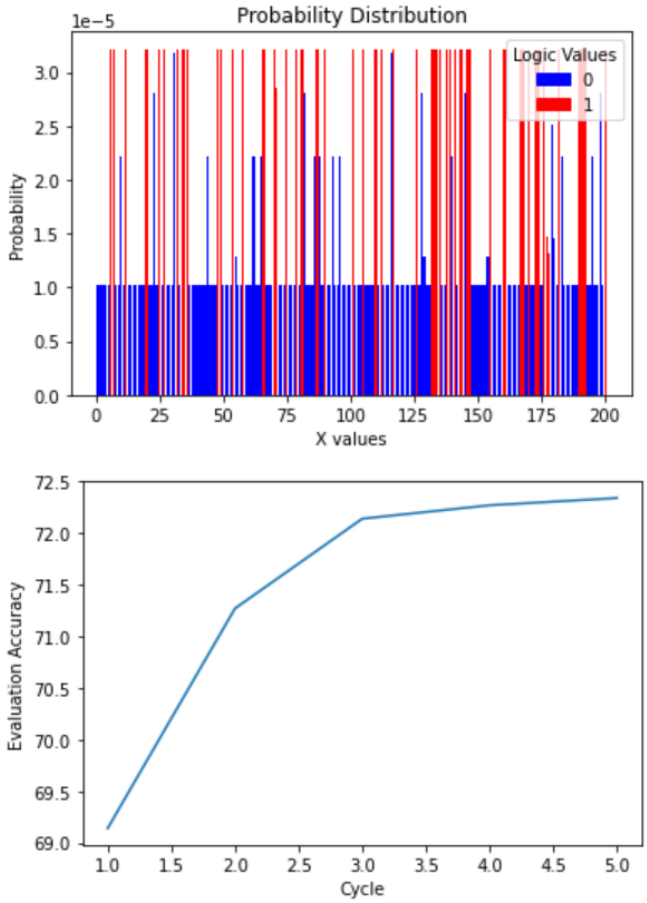

Figure 5.1: Up: final distribution of weights over the first 200 training samples, we can see which errors have been corrected during the Qboost optimization. **Down**: Evaluation accuracy value for the 5 cycles that this Qboost run lasted.

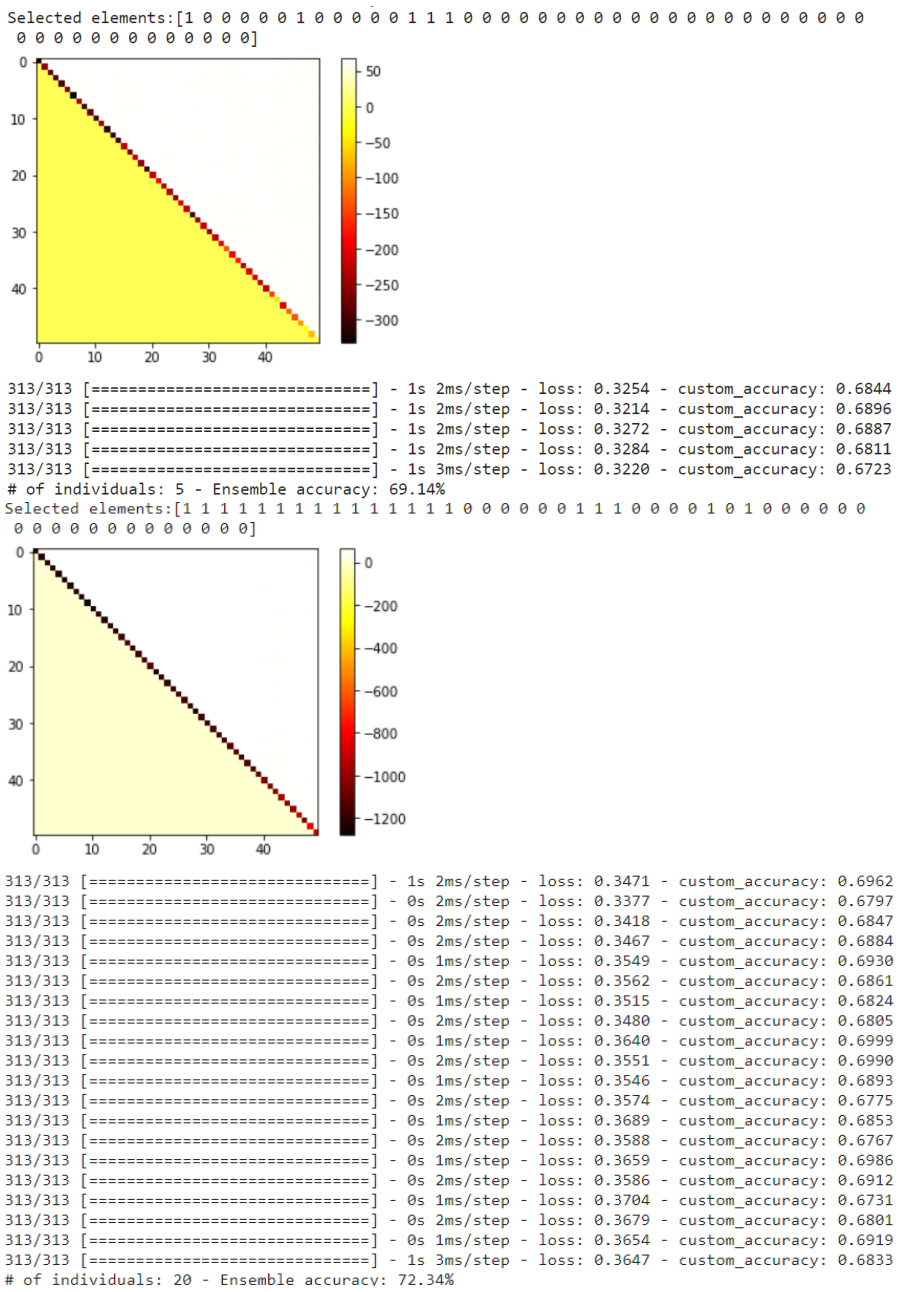

Figure 5.2: Up: Best result obtained during the fisrt optimization run with $\lambda = 2500$; **down**: Best result obtained during the last optimization run with $\lambda = 2100$

Here, our findings largely align with what we anticipated in Section 4.3. Figures 5.1 and 5.2 illustrate that the Qboost procedure incrementally strengthens the ensemble until it reaches the desired final size. Furthermore, Figure 5.2 shows a notable improvement in the ensemble's classification accuracy. Initially, the performance is comparable to the average of individual weak learners, but eventually, it surpasses them by more than three per-

centage points. These results not only confirm the effectiveness of the Qboost algorithm but also reassure us about the accuracy of matrix element computations and the integrity of the overall code implementation.

5.1.2 C10 DATASET

For what concerns the ensemble construction within the C10 data-set several things can be noted from the results reported in this section. In particular from Figure 5.3 it can be appreciated the difference between the two implementations of the Greedy algorithm. The first one that focus in optimizing metrics, that is obtained through a process of optimization and evaluation over the same data-set obtains a significantly higher score if compared both with the Greedy algorithm over the predictions. This is expected and we think that it is due for the most part to overfitting. Moreover, we can clearly see that the "weak boosting" procedure seems to be completely ineffective as we have performed the QBoost optimization building QUBO matrices of dimension 50.

To rule out any implementation errors affecting the theoretical functionality of the boosting procedure, we conducted Qboost optimization with a minimal number of weak learners. his was done to increase the likelihood of observing variations in subsequent optimization steps. Minor fluctuations were observed in the initial stages, but the algorithm eventually converged.

This is reassuring about the correctness of the implementation, however it could imply a low level of heterogeneity among the different learners, or the need to increase the initial pool from which we choose them, $\{h_i\}$.

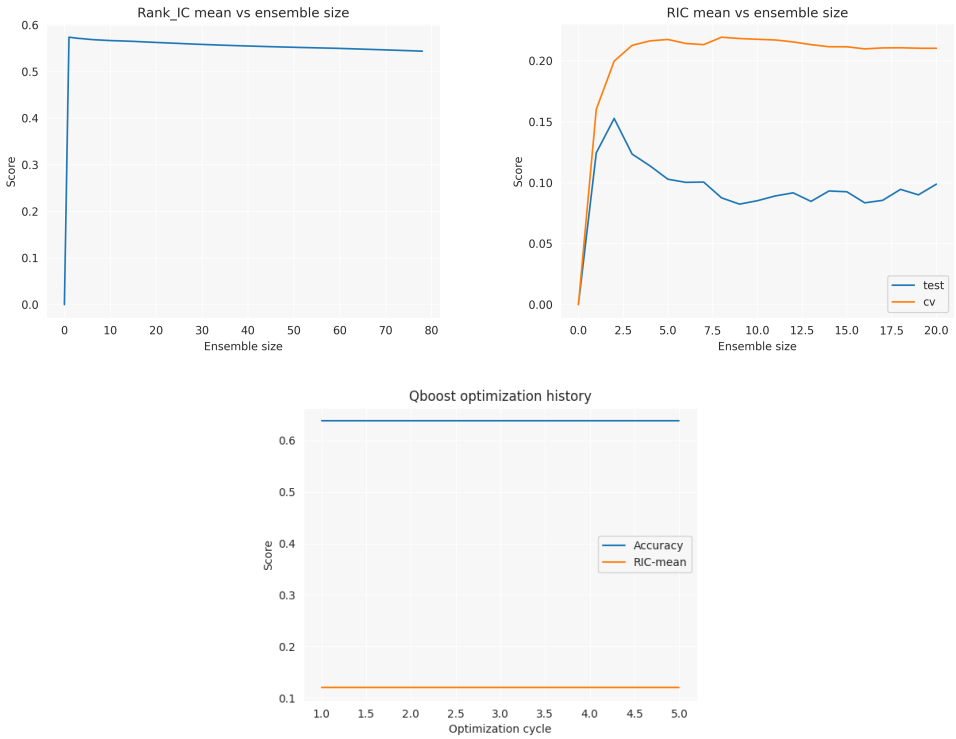

Figure 5.3: Up-left: C10 - ensemble optimization history obtained from Greedy alg. performed over aggregated metrics; **Up-right**: C10 - ensemble optimization history obtained from Greedy alg. performed over the predictions; **down**: C10 - ensemble optimization history obtained from multi-label Qboost implementation

Ensembles obtained from the different procedures posses similar sizes as shown in listings 5.1, 5.2 and 5.3. The numerosity of these ensembles is approximately coherent with the "law of diminishing returns in ensemble construction" outlined in [24]. It states that in general the best number of learners for an ensemble is given by the number of different labels of the classification problem we aim to solve. Interestingly, the two ensembles that resulting from the greedy procedure from predictions and from Qboost shares a weak learner.

```
{
"Selected Learners": [
    691968,
    691981
    ],
"maximize_metric": "mean",
"ensemble_metrics": {
    "mean": 0.5724,
    "sortino": 1.0937,
    "sharpe": 0.5544
    }
}
```

Listing 5.1: C10-greedy metrics

```
{
"Selected Learners": [
    691938,
    692023
    ],
"maximize_metric": "mean",
"ensemble_metrics": {
    "mean": 0.1526,
    "sortino": 0.5510,
    "sharpe": 0.2817
    }
}
```

Listing 5.2: C10 - greedy predictions

```
{
"Selected Learners": [
        691938,
        691982,
        692005
    ],
"metrics": {
    "mean": 0.1198,
    "sortino": 0.3974,
    "sharpe": 0.2106
    }
}
```

Listing 5.3: C10 - Qboost

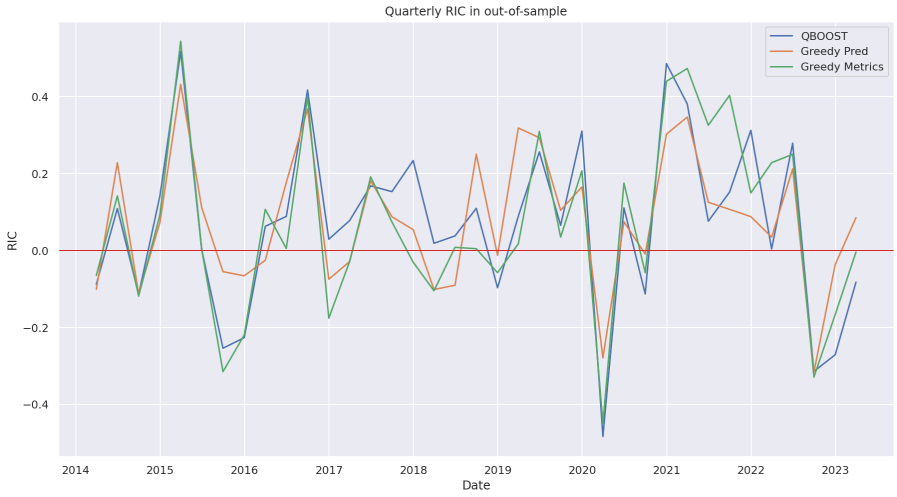

Figure 5.4: Out-of-sample performances of the three ensembles built over C10 data-set

Lastly, in 5.4 and 5.1 are reported the performance of the three ensembles in an OOS period. In the financial context, even a slightly better-than-random performance is considered informative. The performances of the three ensembles are fairly comparable, with the Greedy algorithm based on predictions performing best, followed by Qboost and the Greedy algorithm using aggregated metrics. The consistent reduction in performance for the latter suggests that overfitting is indeed a factor.

	QBOOST	Greedy Pred	Greedy Metrics
RIC mean	0.07195	0.07850	0.06662
RIC std	0.4830	0.4663	0.4895
RIC Down std	0.2569	0.2456	0.2591

Table 5.1: C10 - Summary of RIC metric in OOS period

5.1.3 FTSEMIB v1 data-set

For the FTSEMIB v1 dataset, ensembles were generated using only the Qboost and Greedy algorithms based on predictions, foregoing the Greedy algorithm over metrics due to its tendency to overfit, as observed in the C10 dataset. Qboost optimization has been conducted considering 50 out of 100 weak learners for the QUBO optimization.

Figure 5.5 reveals that the optimization histories for both methods suggest comparable performance, although the Greedy algorithm shows a slight advantage. It's also worth noting that the "weak boosting" procedure proves ineffective here as well.

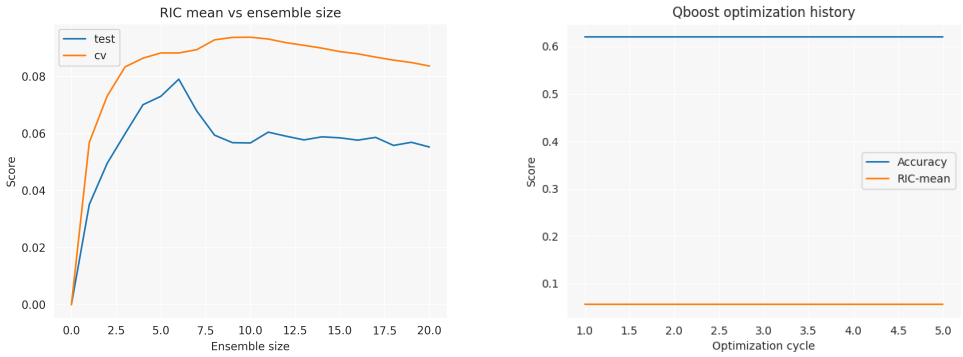

Figure 5.5: Left: FTSEMIB v1 - ensemble optimization history obtained from Greedy alg. performed over the predictions; **Right**: FTSEMIB v1 - ensemble optimization history obtained from multi-label Qboost implementation

In terms of size, the ensembles created with the FTSEMIB v1 data are larger than those derived from the C10 data. The numerosity of the ensembles varies significantly between the two methods, as shown in Listings 5.4 and 5.5. Specifically, the Greedy algorithm based on predictions generates an ensemble comprising 6 weak learners, while the Qboost algorithm results in an ensemble of 15. Intriguingly, all 6 learners selected by the Greedy method are also included in the ensemble formed by our multi-label Qboost algorithm.

```
{
"job_ids": [
    694694,
    694715,
    694717,
    694754,
    694748,
    694737
    ],
"ensemble_metrics": {
    "mean": 0.0791,
    "sortino": 0.6058,
    "sharpe": 0.3042
    }
}
```


Listing 5.4: FTSEMIB v1 - greedy predicitons


```
{
"Selected Agents": [
    694666,694676,694680,
    694684,694694,694695,
    694710,694715,694717,
    694730,694733,694735,
    694737,694748,694754
    ],
"metrics": {
    "mean": 0.0555,
    "sortino": 0.3998,
    "sharpe": 0.2108
    }
}
```


Listing 5.5: FTSEMIB v1 - Qboost

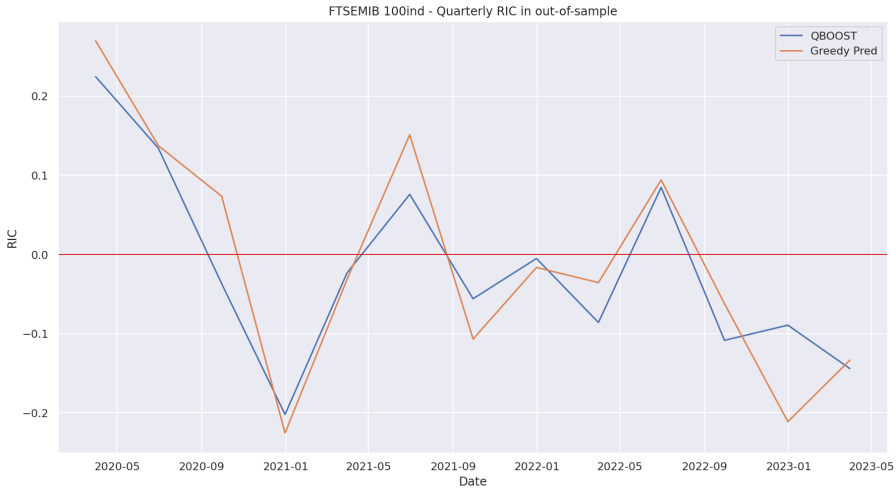

Figure 5.6: Out-of-sample performances of the three ensembles built over FTSEMIB v1 data-set

As was done with the C10 ensembles, we evaluated the FTSEMIB v1 ensembles during an OOS period. The results are depicted in Figure 5.6 and summarized in Table 5.2. In absolute terms, both ensembles perform worse than those generated with the C10 data. Nevertheless, the ensemble created via the Greedy algorithm shows marginally better performance, albeit not significantly so. The behaviour of the two ensembles generated for this dataset are similar both in terms of numerical performance and in terms of the behaviour in the OOS period showed in Figure 5.6.

	QBOOST	Greedy Pred
RIC mean	-0.0192	-0.0088
RIC std	0.2885	0.2998
RIC Down std	0.1675	0.1831

Table 5.2: FTSEMIB v1 - Summary of RIC metric in OOS period

FTSEMIB v2 data-set

The same procedures adopted for FTSEMIB v1 data-set have been applied to FTSEMIB v2, that covers a different time interval and involves the performance of 500 weak learners. The Qboost optimization has been conducted considering 200 agents per cycle.

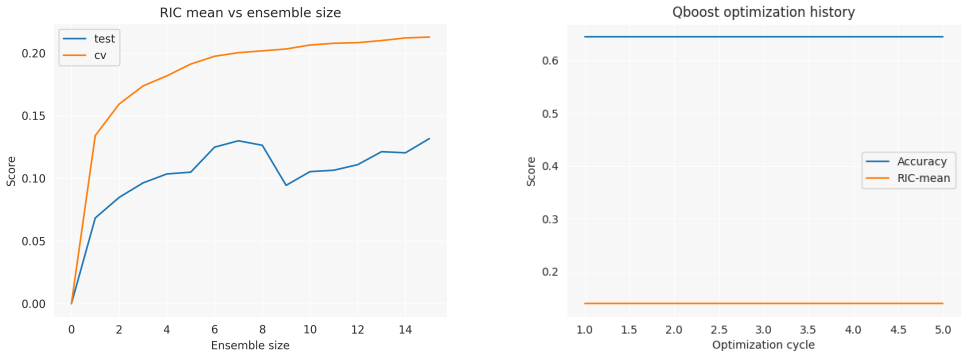

Figure 5.7: Left: FTSEMIB v2 - ensemble optimization history obtained from Greedy alg. performed over the predictions; **Right**: FTSEMIB v2 - ensemble optimization history obtained from multi-label Qboost implementation

Figure 5.7 shows that the performance of Qboost and the greedy algorithm are comparable, although Qboost has a slight edge. However, the composition of the ensembles diverges significantly from the results on FTSEMIB v1. In this case, the greedy algorithm selects a considerably larger ensemble, comprising 15 weak learners, as shown in listing 5.6. On the other hand, Qboost converges to a lean ensemble of just three agents, as detailed in listing 5.7.

```
{
"job_ids": [
    695093,694863,694772,
    694801,694995,694935,
    695064,695141,694700,
    694722,694899,694849,
    695016,694766,694754
    ],
"ensemble_metrics": {
        "mean": 0.1317,
        "sortino": 1.0203,
        "sharpe": 0.4909
    }
}
```


Listing 5.6: FTSEMIB v2 - greedy predicitons


```
{
"Selected Agents": [
    694730,
    694935,
    695005
    ],
"metrics": {
    "mean": 0.1391,
    "sortino": 1.1532,
    "sharpe": 0.5440
    },
}
```


Listing 5.7: FTSEMIB v2 - Qboost

For FTSEMIB v2, the out-of-sample (OOS) results align with those obtained for the v1 dataset. Overall, the performance on the FTSEMIB datasets is not particularly strong. Nonetheless, our primary interest lies in the comparative analysis of the two ensemble methods rather than in their absolute performance. In this regard, Qboost slightly outperforms the greedy algorithm in OOS data, though the difference is not significant.

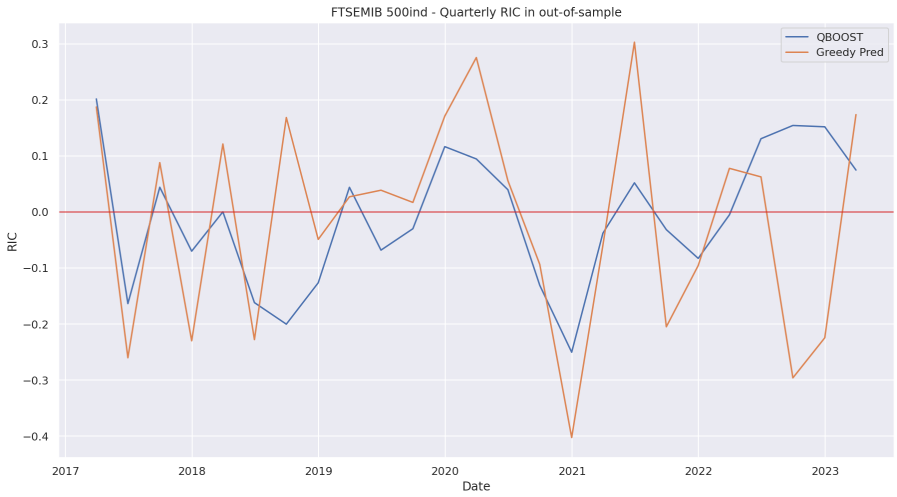

Figure 5.8: Out-of-sample performances of the three ensembles built over FTSEMIB v2 data-set

Lastly, Figure 5.8 reveals more moments where the performance of the two ensembles diverges, compared to what is observed in Figure 5.6 for the FTSEMIB v1 dataset.

As evidenced by Tables 5.2 and 5.3, ensembles generated from the FTSEMIB dataset generally exhibit suboptimal performance during the out-of-sample (OOS) period. While our primary focus is on the comparative performance between the two ensemble methods, it's important to contextualize these findings within the broader

	QBOOST	Greedy Pred
RIC mean	-0.01016	-0.01584
RIC std	0.2194	0.2640
RIC Down std	0.1318	0.1542

Table 5.3: FTSEMIB v2 - Summary of RIC metric in OOS period

landscape of financial modeling. In this domain, even a marginal advantage over random predictions can be considered a success. The poor performance in the OOS period may be attributed to 'concept drift' in the market dynamics. Specifically, the relationships and features that the models have identified and optimized for during the training phase may no longer hold true or be as influential in the OOS period, leading to the observed decline in predictive accuracy.

5.2 QUBO optimization with a fixed size QUBO matrix

Here we present the results of the experiment outlined in 4.4. The experiment is conducted only over the Axyon data-sets. In particular we exploit all the available weak learners for the C10 data-set and for the FTSEMIB(v1) dataset which both are obtained from 100 learners. Instead we exploit 200 of the 500 learners present in FTSEMIB(v2) due to computational time constraints.

C10 data-set

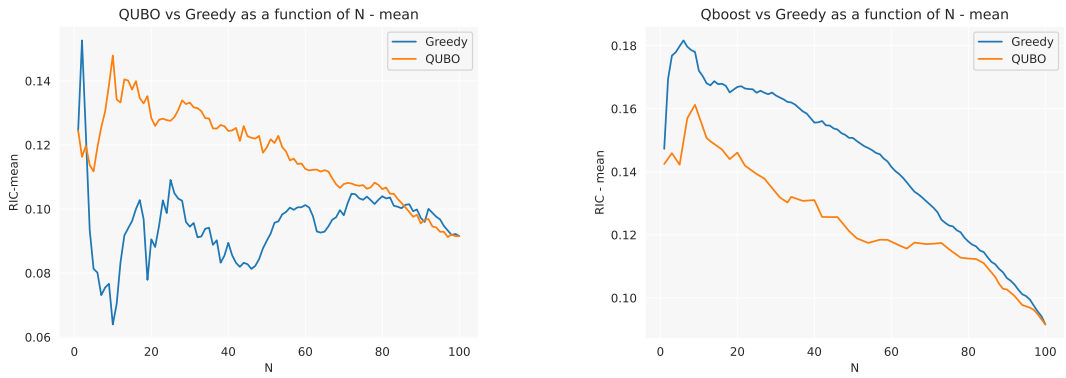

Figure 5.9: C10 data-set - **Left**: Qboost inner cycle vs greedy, optimized over cv data and evaluated over test data; **Right**: Qboost inner cycle vs greedy algorithm, optimized and tested over test data.

FTSEMIB v1 data-set

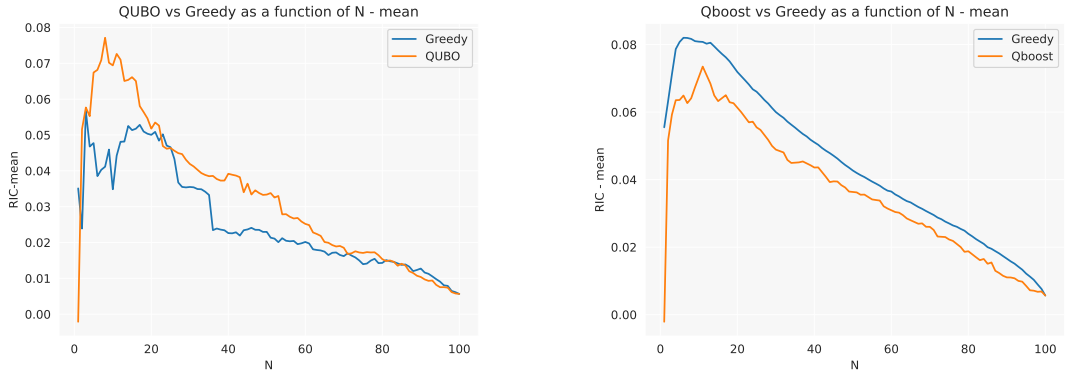

Figure 5.10: FTSEMIB v1 data-set - **Left**: Qboost inner cycle vs greedy , optimized over cv data and evaluated over test data; **Right**: Qboost inner cycle vs greedy algorithm, optimized and tested over test data.

FTSEMIB v2 data-set

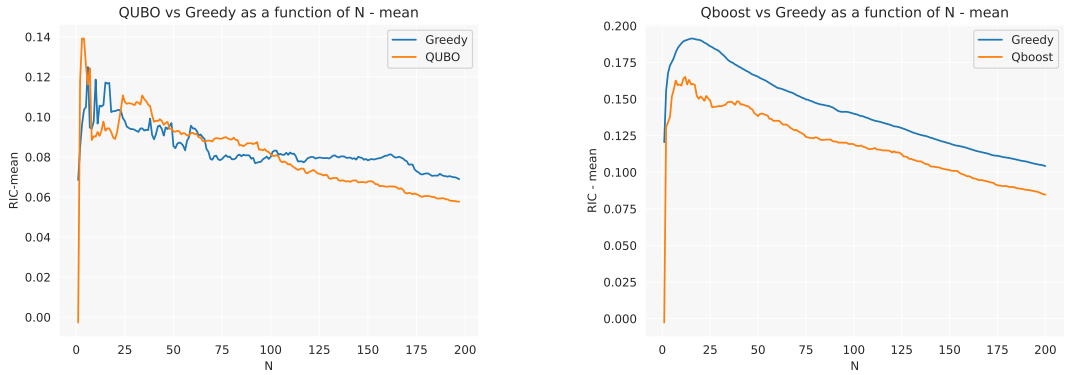

Figure 5.11: FTSEMIB v2 data-set - **Left**: Qboost inner cycle vs greedy , optimized over cv data and evaluated over test data; **Right**: Qboost inner cycle vs greedy algorithm, optimized and tested over test data.

Figures 5.9, 5.10 and 5.11 presents the average RIC metrics obtained from ensembles with different numerosity. As specified in section 4.4 the different ensembles built through QUBO optimization are obtained varying the regularization parameter λ. Instead, the results from the greedy procedure are obtained specifying the desired ensemble size as an hyper-parameter in the algorithm initialization.

From all the experiment we can see that when the ensembles are builded with 'cv' data and evaluated over a different chunk of data exploited as 'test' data, the results obtained from QUBO optimizations clearly outperform greedy in two out of three analyzed cases. However, the situation change when the ensembles are builded and

evaluated over the same data. In this case the performance of the greedy algorithm are consistently better than the one from QUBO optimizations. This last methodology does not represent the standard described by theory when it comes to building and evaluating models in the ML framework. In fact it is exceedingly prone to overfitting, however it is the standard followed in Axyon and therefore it has been reported.

It must be highlighted the fact that while QUBO optimizations are based on the accuracy value of the ensemble about the multi-label classification problem, the greedy algorithm although in its implementation over the predictions does not take into account metrics from aggregated periods of time in the dataset, it is still optimized through the evaluation of these metrics from each data sample. This is the reason why the optimizations curve does not need to start from the same values both for QUBO and greedy optimization procedures: the best weak learner in terms of accuracy is not guaranteed to be the best one also in term of RIC.

On the contrary, the optimization courves must converge to the sam e value for C10 and FTSEMIB v1 datasets. In fact in this case we arrive to consider all the weak learners available, and the performance must be the same. This does not happen for FTSEMIB v2 since we only take into account 200 out of 500 agents.

Finally it can be noted that for all three data-sets taken into account the SA solver does not produce different solutions with the same numerosity.

In Appendix.B are reported the same plots with sortino and shape metrics as well as the parameters with which these results has been obtained.

5.3 QUBO optimization with a variable size QUBO matrix

The following results are based on the experiments detailed in section 4.5. The elements plotted are derived as follows:

- **QUBO**: The orange line on the plots shows the best outcomes from a complete Qboost inner cycle, spanning QUBO matrix dimensions between 2 and 100. Each data point represents the optimal result for that specific dimension.

- **Greedy**: Represented by the blue line, this algorithm iterates over predictions. Each point corresponds to the optimal result from a greedy optimization cycle, where the maximum ensemble size is equal to the current QUBO matrix dimension.

- **Brute Force with Regularization**: The dashed green line indicates results obtained through brute-force optimization of matrices built with a regularization parameter λ equal to the one that produced the best result during SA optimization.

- **Brute Force without Regularization**: The red line presents the outcomes of brute-force optimization without regularization, providing insight into the impact of regularization on performance.

C10 data-set

Figure 5.12: C10 data-set - **Left**: Qboost inner cycle vs greedy - QUBO matrices of variable size - optimized over cv data and evaluated over test data; **Right**: Qboost inner cycle vs greedy algorithm - QUBO matrices of variable size - optimized and evaluated over test data.

FTSEMIB v1 data-set

Figure 5.13: FTSEMIB v1 data-set - **Left**: Qboost inner cycle vs greedy - QUBO matrices of variable size - optimized over cv data and evaluated over test data; **Right**:Qboost inner cycle vs greedy - QUBO matrices of variable size - optimized and tested over test data.

As anticipated, the Greedy algorithm shows a consistent, monotonic increase in performance across both the C10 and FTSEMIB v1 datasets. Conversely, fluctuations are noted in the performance of the Simulated Annealing (SA) algorithm, which is expected given that we are not directly targeting the metric plotted and there is no strict proportionality between RIC and multi-label classification accuracy of the ensemble.

Both Figures 5.12 and 5.13 align with results discussed in section 5.2. The highest value in the plots from that section should correspond to the 100^{th} data point in this section's plots, provided the regularization param-

eter schedule is sufficiently dense. Unlike the manual estimation of λ in section 5.2, this experiment required an automated approach for defining it. Specifically, we searched for two λ values that produced ensembles with a minimum of 2 members and a maximum equal to the size of the QUBO matrix.

The

Lastly, additional plots concerning the RIC Sortino ratio and RIC Sharpe ratio can be found in Appendix B.

6

Conclusions and future prospects

This thesis consists in a study of the Quadratic Unconstrained Binary Optimization Problem environment both from a theoretical and from a practical point of view, with particular attention in the declination of QUBO as a resource useful to describe the problem of building an ensemble of Machine Learning models.

It serves as a starting point in the attempt to improve the ensemble building step inside Axyon ML pipeline. The approach adopted in this work is completely different from the one exploited at Axyon as it approaches the construction of the ensemble as a pure combinatorial problem.

In fact, the aim was to introduce a procedure able to explore the space of possible configurations of weak learners in a more extended and more efficient way than that of the greedy algorithm with which we confronted.

However, from the experimental results it is still unclear if the adoption of such an approach could provide substantial and consistent benefits. Further research are needed to establish this with certainty. In particular, several observations from the study point toward promising directions for future research.

A critical observation is that diversity among weak learners is not sufficient or, if sufficient, not adequately exploited. One solution could be to devise a QUBO that involves terms that quantify diversity and reward it accordingly [8].

An aspect that surely should be taken into account is the computational complexity of the QUBO we have defined, with this respect a study about methods that helps to reduce the number of variable needed to define our QUBO (shrinkage) could be beneficial [25].

Moreover, future trials that involves QUBO problems that takes into account directly the rankings produced with the learners predictions remains interesting, besides the attempt that has been made with the Covariance formulation. In fact, some works that involves financial metrics in QUBO definition have been already published [26].

Finally, the implementation created during this work of thesis will be adapted to produce QUBO representations that can be solved through Quantum or Quantum-hybrid solvers through the algorithm reported in the introductory section of this work.

References

[1] H. N. V. S. D. G. R. W. G. Macready, "Qboost: Large scale classifier training with adiabatic quantum optimization," *Journal of Machine Learning Research*, no. 25, pp. 333–348„ 2012.

[2] H. Neven, V. S. Denchev, G. Rose, and W. G. Macready, "Training a binary classifier with the quantum adiabatic algorithm," 2008.

[3] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[4] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[5] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.

[6] D. Morrison, R. Wang, and L. C. De Silva, "Ensemble methods for spoken emotion recognition in call-centres," *Speech Communication*, vol. 49, no. 2, pp. 98–112, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167639306001713

[7] I. K. Nti, A. F. Adekoya, and B. A. Weyori, "A comprehensive evaluation of ensemble learning for stock-market prediction," *Journal of Big Data*, vol. 7, no. 1, p. 20, 2020. [Online]. Available: https://doi.org/10.1186/s40537-020-00299-5

[8] G. Brown, J. Wyatt, R. Harris, and X. Yao, "Diversity creation methods: a survey and categorisation," *Information Fusion*, vol. 6, no. 1, pp. 5–20, 2005, diversity in Multiple Classifier Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253504000375

[9] T. K. Ho, "Random decision forests," in *Proceedings of 3 rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282 vol.1.

[10] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[11] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[12] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang, "The unconstrained binary quadratic programming problem: A survey," *Journal of Combinatorial Optimization*, vol. 28, 07 2014.

[13] A. Lucas, "Ising formulations of many NP problems," *Frontiers in Physics*, vol. 2, 2014. [Online]. Available: https://doi.org/10.3389%2Ffphy.2014.00005

[14] H. Yasuoka, "Computational complexity of quadratic unconstrained binary optimization," *CoRR*, vol. abs/2109.10048, 2021. [Online]. Available: https://arxiv.org/abs/2109.10048

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: http://www.jstor.org/stable/1690046

[16] T. Albash and D. A. Lidar, "Adiabatic quantum computation," *Reviews of Modern Physics*, vol. 90, no. 1, jan 2018. [Online]. Available: https://doi.org/10.1103%2Frevmodphys.90.015002

[17] ——, "Demonstration of a scaling advantage for a quantum annealer over simulated annealing," *Phys. Rev. X*, vol. 8, p. 031016, Jul 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.8.031016

[18] H. Neven, V. S. Denchev, G. Rose, and W. G. Macready, "Training a large scale classifier with the quantum adiabatic algorithm," 12 2009. [Online]. Available: https://arxiv.org/pdf/0912.0779.pdf

[19] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd International Conference on Machine Learning*. Bonn, Germany: Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399, 2005.

[20] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," vol. 227, 06 2007, pp. 129–136.

[21] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," 07 2004.

[22] R. Caruana, A. Munson, and A. Niculescu-Mizil, "Getting the most out of ensemble selection," 12 2006, pp. 828–833.

[23] L. Leclerc, L. Ortiz-Guitierrez, S. Grijalva, B. Albrecht, J. R. K. Cline, V. E. Elfving, A. Signoles, L. Henriet, G. D. Bimbo, U. A. Sheikh, M. Shah, L. Andrea, F. Ishtiaq, A. Duarte, S. Mugel, I. Caceres, M. Kurek, R. Orus, A. Seddik, O. Hammammi, H. Isselnane, and D. M'tamon, "Financial risk management on a neutral atom quantum processor," 12 2022. [Online]. Available: https://arxiv.org/pdf/2212.03223.pdf

[24] H. R. Bonab and F. Can, "Less is more: A comprehensive framework for the number of components of ensemble classifiers," *CoRR*, vol. abs/1709.02925, 2017. [Online]. Available: http://arxiv.org/abs/1709.02925

[25] M. Lewis and F. Glover, "Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis," 2017.

[26] M. Mattesi, L. Asproni, C. Mattia, S. Tufano, G. Ranieri, D. Caputo, and D. Corbelletto, "Financial portfolio optimization: a qubo formulation for sharpe ratio maximization," 2023.

A

Covariance QUBO matrix elements

$$
\begin{aligned}
Q_{ii} = -\sum_{s=1}^{S}\frac{1}{P_s^2}\Bigg[\sum_{k=1}^{P_s}\sum_{z=1}^{P_s}\Bigg(&\frac{1}{N^2}h_{i,s,k}h_{i,s,z}y_{s,k}y_{s,z} - \frac{P_s+1}{2N^2P_s}h_{i,s,k}h_{i,s,z}y_{s,k} - \frac{y_{s,k}y_{s,z}}{N^2P_s}\sum_{v=1}^{P_s}h_{i,s,k}h_{i,s,v} + \\
&+ \frac{P_s+1}{2N^2P_s^2}y_{s,k}\sum_{v=1}^{P_s}h_{i,s,k}h_{i,s,v} - \frac{P_s+1}{2N^2P_s}h_{i,s,k}h_{i,s,z}y_{s,z} + \frac{(P_s+1)^2}{4N^2P_s^2}h_{i,s,k}h_{i,s,z} + \frac{P_s+1}{2N^2P_s^2}y_{s,z}\sum_{v=1}^{P_s}h_{i,s,k}h_{i,s,v} - \\
&- \frac{(P_s+1)^2}{4N^2P_s^3}\sum_{v=1}^{P_s}h_{i,s,k}h_{i,s,v} - \frac{y_{s,k}y_{s,z}}{N^2P_s}\sum_{v=1}^{P_s}h_{i,s,z}h_{i,s,v} + y_{s,k}\frac{P_s+1}{2N^2P_s^2}\sum_{v=1}^{P_s}h_{i,s,z}h_{i,s,v} + \frac{y_{s,k}y_{s,z}}{P_s^2N^2}\sum_{v=1}^{P_s}\sum_{t=1}^{P_s}h_{i,s,v}h_{i,s,t} - \\
&- \frac{P_s+1}{2N^2P_s^3}y_{s,k}\sum_{v=1}^{P_s}\sum_{t=1}^{P_s}h_{i,s,v}h_{i,s,t} + y_{s,z}\frac{P_s+1}{2N^2P_s^2}\sum_{v=1}^{P_s}h_{i,s,z}h_{i,s,v} - \frac{(P_s+1)^2}{4N^2P_s^3}\sum_{v=1}^{P_s}h_{i,s,z}h_{i,s,v} - \\
&- \frac{P_s+1}{2N^2P_s^3}y_{s,z}\sum_{v=1}^{P_s}\sum_{t=1}^{P_s}h_{i,s,v}h_{i,s,t} + \frac{(P_s+1)^2}{4N^2P_s^4}\sum_{v=1}^{P_s}\sum_{t=1}^{P_s}h_{i,s,v}h_{i,s,t}\Bigg)\Bigg] + \lambda
\end{aligned}
\tag{A.1}
$$

$$
\begin{aligned}
Q_{ij} = -\sum_{s=1}^{S}\frac{1}{P_s^2}\Bigg[\sum_{k=1}^{P_s}\sum_{z=1}^{P_s}\Bigg(&\frac{1}{N^2}h_{i,s,k}h_{i,s,z}y_{s,k}y_{s,z} - \frac{P_s+1}{2N^2P_s}h_{i,s,k}h_{i,s,z}y_{s,k} - \frac{y_{s,k}y_{s,z}}{N^2P_s}\sum_{v=1}^{P_s}h_{i,s,k}h_{i,s,v} + \\
&+ \frac{P_s+1}{2N^2P_s^2}y_{s,k}\sum_{v=1}^{P_s}h_{i,s,k}h_{i,s,v} - \frac{P_s+1}{2N^2P_s}h_{i,s,k}h_{i,s,z}y_{s,z} + \frac{(P_s+1)^2}{4N^2P_s^2}h_{i,s,k}h_{i,s,z} + \frac{P_s+1}{2N^2P_s^2}y_{s,z}\sum_{v=1}^{P_s}h_{i,s,k}h_{i,s,v} - \\
&- \frac{(P_s+1)^2}{4N^2P_s^3}\sum_{v=1}^{P_s}h_{i,s,k}h_{i,s,v} - \frac{y_{s,k}y_{s,z}}{N^2P_s}\sum_{v=1}^{P_s}h_{i,s,z}h_{i,s,v} + y_{s,k}\frac{P_s+1}{2N^2P_s^2}\sum_{v=1}^{P_s}h_{i,s,z}h_{i,s,v} + \frac{y_{s,k}y_{s,z}}{P_s^2N^2}\sum_{v=1}^{P_s}\sum_{t=1}^{P_s}h_{i,s,v}h_{i,s,t} - \\
&- \frac{P_s+1}{2N^2P_s^3}y_{s,k}\sum_{v=1}^{P_s}\sum_{t=1}^{P_s}h_{i,s,v}h_{i,s,t} + y_{s,z}\frac{P_s+1}{2N^2P_s^2}\sum_{v=1}^{P_s}h_{i,s,z}h_{i,s,v} - \frac{(P_s+1)^2}{4N^2P_s^3}\sum_{v=1}^{P_s}h_{i,s,z}h_{i,s,v} - \\
&- \frac{P_s+1}{2N^2P_s^3}y_{s,z}\sum_{v=1}^{P_s}\sum_{t=1}^{P_s}h_{i,s,v}h_{i,s,t} + \frac{(P_s+1)^2}{4N^2P_s^4}\sum_{v=1}^{P_s}\sum_{t=1}^{P_s}h_{i,s,v}h_{i,s,t}\Bigg)\Bigg]
\end{aligned}
\tag{A.2}
$$

B

Results visualization

B.1 Fixed size QUBO matrix - variable λ

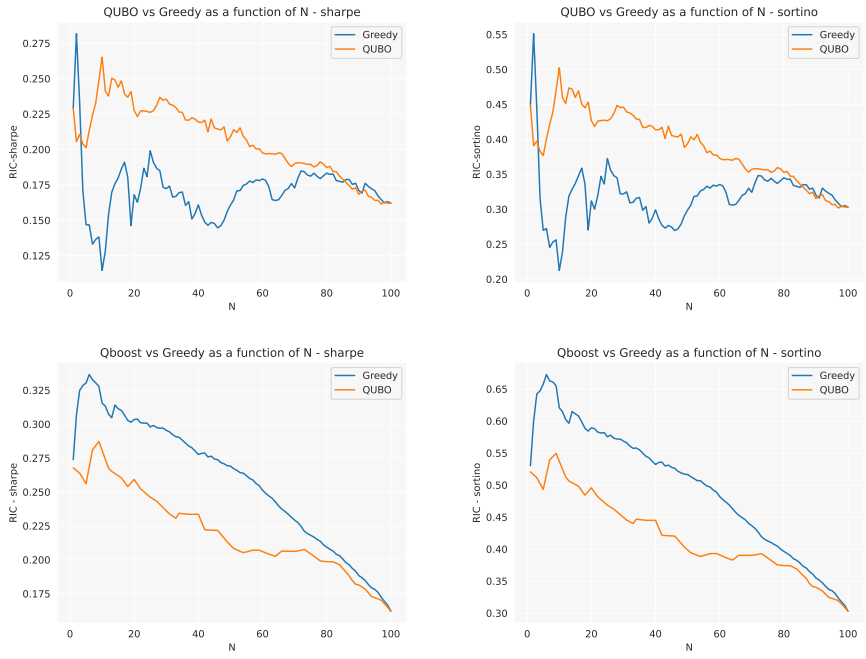

Figure B.1: C10 data-set - Experiment outlined in Sec. 4.4 sharpe and sortino ratios

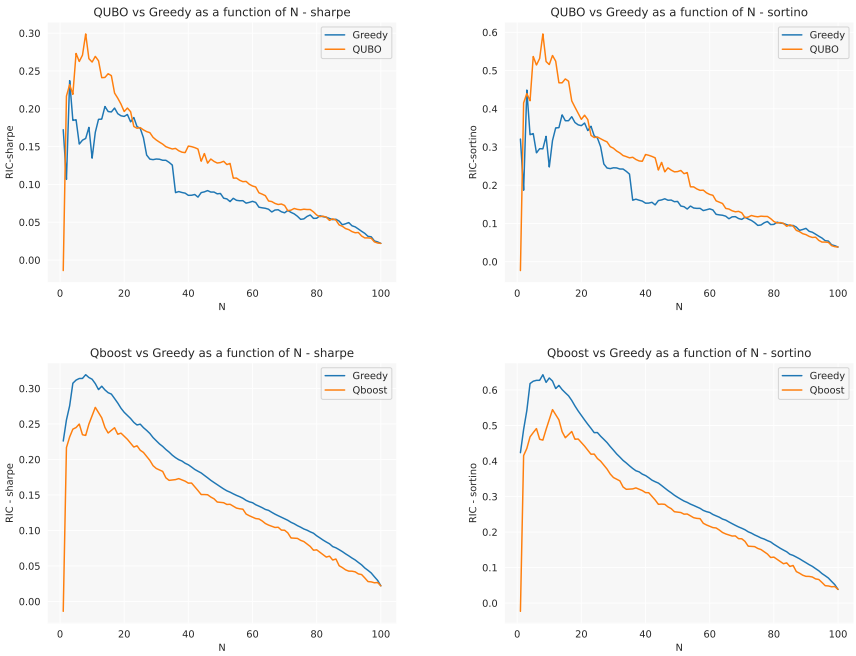

Figure B.2: FTSEMIB v1 data-set - Experiment outlined in Sec. 4.4 sharpe and sortino ratios

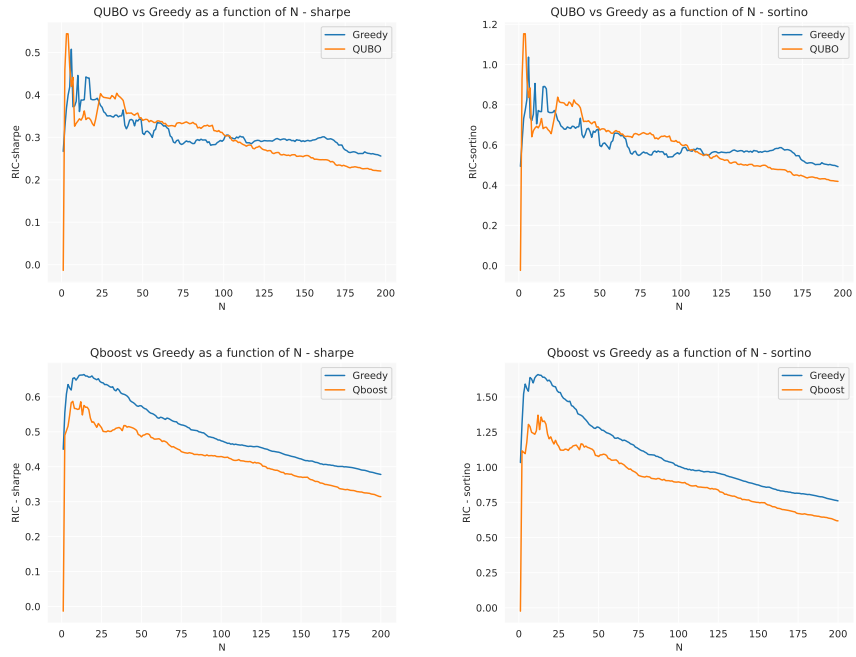

Figure B.3: FTSEMIB v2 data-set - Experiment outlined in Sec. 4.4 sharpe and sortino ratios

B.2 Variable size QUBO matrix - variable λ

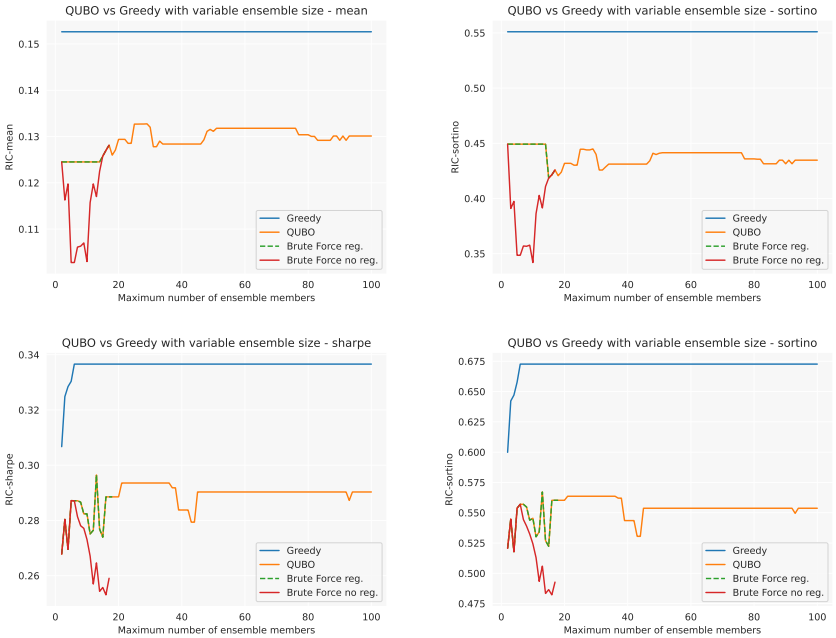

Figure B.4: C10 data-set- Experiment outlined in Sec. 4.5 sharpe and sortino ratios

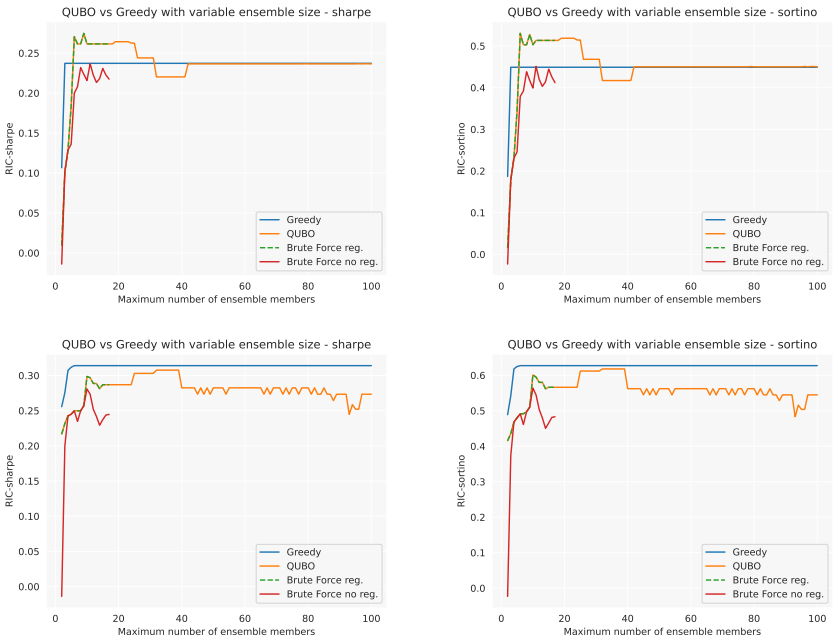

Figure B.5: FTSEMIB v1 data-set - Experiment outlined in Sec. 4.5 sharpe and sortino ratios

Acknowledgments

I wish to express my sincere credit to Professor Marco Baiesi for graciously accepting the role of my academic advisor and providing guidance during this stage of my academic journey.

My appreciation also extends to the entire team at Axyon AI, who have welcomed me warmly and made me feel a part of their community from the outset. Particular thanks go to my advisor at Axyon AI, Giovanni Davoli, for his continuous assistance and valuable insights. I would also like to recognize Jacopo Credi from Axyon and Dr. Davide Venturelli from NASA for their consistent support, advice, enriching conversations, and the trust they have placed in me since the beginning of my experience at Axyon.

I would like to extend my heartfelt gratitude to a number of individuals who have made invaluable contributions to my personal and academic adventure. Foremost, I owe a deep debt of gratitude to my family and friends for their unwavering support throughout this endeavor. A special acknowledgment is due to my girlfriend, Alessandra, who has been a constant companion in both light-hearted and serious moments, offering me an immeasurable source of strength, courage and inspiration.