Machine Learning Execution Time in Asset Pricing

Umit Demirbaga[‡]

Yue Xu[§]

This version: November 2, 2023

[‡]Department of Medicine, University of Cambridge; European Bioinformatics Institute (EMBL- EBI); Department of Computer Engineering, Bartin University. Email: ud220@cam.ac.uk.

[§]Department of Finance, Business School, Durham University. Email: xu.yue@durham.ac.uk.

Machine Learning Execution Time in Asset Pricing

Abstract

In the fast-paced world of finance, where timely decisions can yield substantial gains or losses, machine learning models with time-consuming training and prediction may miss crucial market timing opportunities. This study examines the machine learning model execution time including both training and prediction phases, in empirical asset pricing. We conduct a comprehensive analysis of machine learning execution time, examining ten models and introducing two strategies to save time: feature reduction and the reduction of time observations. Our findings reveal that XGBoost stands out as a top performer, demonstrating relatively low execution times compared to other machine learning models, with exceptional accuracy, boasting an out-of-sample R^2 of 0.78 and a Sharpe ratio of 1.76. Furthermore, feature reduction and shorter time observations reduce execution time by as much as 18 times while also slightly enhancing investment performance. This research underscores the vital interplay between model accuracy and execution time to make accurate and prompt investment decisions in practice.

Keywords: Execution Time; Asset Pricing; Machine Learning; XGBoost; Investment Performance

JEL Classification: C52; C55; C58; G12; G17

This version: November 2, 2023.

I. Introduction

Recent studies highlight the improved precision in return predictability when incorporating machine learning methods into financial predictions (Gu et al. (2020); Bianchi et al. (2021); Bali et al. (2021)). While accuracy has often been the primary focus in evaluating machine learning models in asset pricing, an equally critical factor that deserves attention is the machine learning model execution time. The concept of execution time, in the context of machine learning implementation, encompasses the temporal duration required to complete the entire lifecycle of a machine learning model. This includes critical phases, including model training, hyperparameter optimization, and evaluation during testing. To facilitate comprehension, we split this time into two main parts: training time and prediction time.

Time is an invaluable resource in the fast-paced world of finance, where timely decisions can lead to substantial gains or losses. The rapid evolution of financial markets demands real-time responses, and machine learning models that consume excessive time for training and prediction might miss critical **market timing opportunities**. Machine learning models that exhibit prolonged times may compromise the timeliness of decision-making, rendering their valuable insights less actionable.

Beyond market timing, the execution time of machine learning models carries economic implications. **The cost of time** encompasses multiple dimensions, including labor costs for analysts and the expenses associated with running resource-intensive computations. For example, high-performance computing (HPC) services often entail charges based on processing time, making efficient algorithms financially prudent. By optimizing the time, financial institutions can mitigate the financial burden associated with prolonged computation, ensuring that computational resources are used effectively.

There are a large number of characteristics in empirical asset pricing, as is common in measuring equity risk premiums (e.g., De Bondt and Thaler (1985); Fama and French (1992); Jegadeesh and Titman (1993); Amihud (2001); Ang et al. (2006); Daniel and Titman

(2006)). The high dimensionality of financial data can amplify the computational burden, potentially leading to resource-intensive processes that hinder real-time decision-making. By accounting for machine learning execution times, researchers and practitioners can strike a balance among accuracy, efficiency, and profitability, ultimately creating models that generate valuable predictions while responding swiftly to evolving market dynamics.

This paper conducts a comprehensive evaluation of machine learning execution time in empirical asset pricing. We conduct a comparable analysis of execution time for a diverse set of machine learning models. Additionally, we investigate two strategies to reduce execution time: first, by reducing the number of characteristics, and second, by reducing training and prediction time observations.

Given the extensive list of stock-level characteristics and the prolonged time periods, we use a high-performance computing (HPC) cluster boasting 64 CPU cores and 1 TB of memory to execute machine learning algorithms and estimate execution times. To ensure robustness and statistical reliability, each algorithm is executed five times, and the resulting execution times is subjected to statistical analysis using standard deviations. This rigorous approach provides insights into the consistency and variability of execution times, shedding light on the computational efficiency and stability of these algorithms in return predictability.

We implement ten distinct machine learning algorithms, using a comprehensive set of monthly stock-level characteristics. Among these algorithms, XGBoost exhibits the highest predictive performance, as evident from its superior out-of-sample R^2 and mean squared errors. Additionally, the long-short portfolio constructed using XGBoost demonstrates the highest Sharpe ratio, attaining a remarkable 1.76. While linear, lasso, and ridge regressions exhibit the shortest training times, each requiring less than 0.4 seconds, XGBoost also showcases a relatively efficient training time of 26.51 seconds, significantly outperforming random forest (1221.14 seconds) and gradient boosting (270.49 seconds). In terms of prediction time, XGBoost is similar to regression models. These results highlight the primary tradeoff between investment performance and training time lies between XGBoost and regression models.

However, despite regressions' advantage in training time, their lower predictive accuracy and inferior investment performance suggest that regressions may not be the ideal choice in real investment decisions.

We narrow our attention to XGBoost as our primary machine learning model due to its superior predictive accuracy and investment performance compared to other models. To further enhance the time efficiency of machine learning algorithms, we employ our first method: reducing the number of characteristics based on their importance in prediction. Notably, characteristics such as the 52-week high, supplier momentum, maximum return over a month, and idiosyncratic risk are identified as having the highest importance, while others like unexpected R&D increases and spinoffs are found unimportant and consequently removed from consideration.

Our feature reduction approach reveals that training time and prediction time exhibit a consistent decrease after removing less important characteristics. Remarkably, as we reduce the number of characteristics to only 31 (with importance exceeding 0.5), training time decreases approximately fivefold, while prediction times decrease around fourfold. In particular, this reduction in characteristics also improves the investment performance, highlighting the significance of identifying crucial stock-level characteristics for return predictability. This not only contributes to save machine learning execution time but also enhances investment performance.

In line with existing literature on feature selection (Kelly et al. (2019); Feng et al. (2020); Freyberger et al. (2020); Kozak et al. (2020)), our approach emphasizes the importance of feature reduction. However, our motivation is distinct; we prioritize feature reduction not only to enhance machine learning execution time but also to augment investment performance.

Furthermore, we investigate the reduction of time observations to enhance machine learning time efficiency. Our findings illustrate that employing the most recent four years of monthly data observations results in a remarkable 18-fold reduction in training time and a threefold reduction in prediction times compared to using a historical dataset spanning six decades. Interestingly, the Sharpe ratio exhibits a slight improvement when employing recent four-year observations in contrast to the longer 60-year dataset. This suggests that data spanning six decades may not hold significant practical value for training machine learning models. Instead, shorter time periods prove to be more informative, indicating that financial practitioners should consider training models within shorter time frames, enabling timely predictions and responses to evolving market dynamics.

This paper makes a substantial contribution to the literature on asset pricing (e.g., Fama and French (2008); Hou et al. (2018)) and the application of machine learning techniques in finance (e.g., Bryzgalova et al. (2019); van Binsbergen et al. (2020); Israel et al. (2020)). There are also some recent studies on machine learning in asset pricing. In particular, Gu et al. (2020) examine machine learning methodologies in empirical asset pricing and demonstrated their superior predictive performance when compared to traditional linear models. Bianchi et al. (2021) investigate the predictability of bond returns and found that extreme trees and neural networks exhibit exceptional forecasting capabilities in this context. Furthermore, Bali et al. (2021) illustrate the advantages of nonlinear machine learning models in predicting option returns and also highlight the enhanced predictive accuracy achieved by incorporating both option and stock characteristics. However, these studies have commonly overlooked the critical dimension of time in the training and prediction phases, thereby limiting the practical applicability of their findings. In this regard, our research stands as a pioneering effort by emphasizing the importance of execution time considerations in implementing prediction models.

Our finding reveals that the XGBoost machine learning model exhibits a distinct advantage in terms of execution time without compromising predictive accuracy. Moreover, we introduce two efficient methods aimed at time-saving: eliminating redundant characteristics and reducing the time observations in the training dataset. Our primary contribution is to document the significance of execution time as the financial landscape increasingly harnesses machine learning's potential for predicting asset prices. While accuracy remains paramount, a model's ability to deliver timely forecasts can significantly influence its practical applicability and impact on decision-making. By factoring in the training and prediction times of machine learning models, researchers and practitioners can find the most appropriate model that unlock actionable insights while navigating the intricacies of real-time decision-making.

The remainder of this paper is structured as follows. Section II provides an overview of our methodology and details about the data used. Section III presents the findings related to machine learning execution times using different models. Section IV analyzes the the impact of different numbers of characteristics on execution time. Section V shows the execution time using different time observations. Section VI offers concluding remarks. Additional details and information can be found in the Appendix.

II. Data And Methodology

II.1. Data

The monthly equity returns are obtained from the Center for Research in Security Prices (CRSP) database. We include all U.S. firms that are listed on the NYSE, AMEX, and NASDAQ. The sample period spans from March 1957 to December 2022, covering 65 years. We use the 1-month Treasury-bill rate from CRSP as a proxy for the risk-free rate, from which we compute the equity excess returns.

We use 209 firm-level predictive characteristics as features to predict stock returns, which include almost all signals in the cross-sectional stock returns literature.[1]

II.2. Handling Missing Value

A significant number of observations are missing in the dataset of firm-level characteristics. Handling missing values is an essential data preprocessing procedure to address the absence of information within a dataset, which involves systematically identifying and replacing missing

[1]The 209 firm-level characteristics are constructed by Chen and Zimmermann (2022) and are available from https://www.openassetpricing.com/data.

data points with appropriate values to maintain the dataset's completeness and integrity (Emmanuel et al. (2021)).

We deployed the k-Nearest Neighbors (k-NN) imputation technique to handle the missing values in the dataset. This approach is based on proximity-based imputation, whereby missing values in a multi-dimensional feature space are estimated using data from their closest neighbours. This technique is chosen for its ability to capture the underlying structure and relationships within the dataset, particularly when the missingness mechanism is believed to be not completely random (Jönsson and Wohlin (2004)). By applying k-NN imputation, we aim to minimize the impact of missing data on the subsequent machine learning algorithms' performance, which ensures that the imputed values are derived from a coherent and data-driven estimation process, thereby preserving the integrity of the finance dataset for subsequent analysis.

II.3. Sample splitting

We split the dataset into two principal subsets: a training set and a testing set. We employ the Pareto Principle, the 80/20 rule, to determine the ratio between the training and testing sets (Sanders (1987)). The training set, equal to 80% of the whole data set, forms the cornerstone of model training, allowing ML algorithms to understand the underlying data patterns and relationships deeply. Conversely, the testing set, the rest of the data, 20%, concealed from the models during training, is important to evaluate model performance and to make accurate predictions out-of-sample. We measure an ML model's predictive power by using the test dataset to construct out-of-sample R_2 and mean squared error.

II.4. Machine Learning Execution Time

Execution time in the machine learning implementation quantifies the computational resources and efficiency demanded by the machine learning algorithm when confronted with a specific dataset and task (Miu and Missier (2012)). In this study, our primary objective is to examine the temporal aspects of machine learning execution, covering both the training and testing phases. We measure the training time and prediction time for each machine learning model.

For robustness and statistical reliability, we executed each algorithm five times, analyzing the resulting execution times and calculating standard deviations. This rigorous approach allowed us to assess the consistency and variability in execution times, providing valuable insights into the computational efficiency and stability of these algorithms.

II.5. Method

Let denote $R_{i,t+1}$ denote the excess return of an asset i at $t+1$,

$$R_{i,t+1} = E_t(R_{i,t+1}) + \epsilon_{i,t+1}, \tag{1}$$

where

$$E_t(R_{i,t+1}) = g^*(z_{i,t}). \tag{2}$$

Our goal is to construct a representation of $E_t(R_{i,t+1})$ using input variables that optimise its ability to explain realized $R_{i,t+1}$ out of sample. Let $z_{i,t}$ denote the N-dimensional vector of stock-level characteristics (input variables). We assume that the conditional expected return denoted as $g^*(.)$ is a function of these characteristics. The following sections will briefly outline each machine learning model's function.

II.6. Linear Regression, Lasso Regression, and Ridge Regression

II.6.1. Linear Regression

Linear regression (LR) is a supervised and widely utilized machine learning algorithm deployed in different areas for modelling and predicting quantitative relationships between variables. This algorithm has gained prominence due to its simplicity, interpretability, and computational efficiency. LR characterizes the relationship between a dependent variable (target) and one or more independent variables (features) by fitting a linear equation to the observed data.

Mathematically, it can be expressed as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \varepsilon, \tag{3}$$

where Y represents the dependent variable, the quantity we aim to predict based on certain factors. The intercept term, β_0, signifies the baseline value of Y when all independent variables $(X_1, X_2, \ldots, X_n)$ are zero, establishing the starting point of the linear relationship. The coefficients $(\beta_1, \beta_2, \ldots, \beta_n)$ are associated with the independent variables and measure how much Y changes for a unit change in each respective X variable while keeping others constant. These coefficients reveal the strength and direction of the relationships. $X_1, X_2, \ldots, X_n$ are the independent variables, the features or attributes influencing Y. Lastly, ε denotes the error term, representing unexplained variability and the difference between the model's predictions and actual observations of Y.

LR has several benefits for financial modelling. Its simplicity and ease of interpretation are its main advantages. Financial analysts can evaluate the influence of each independent variable on the dependent variable using the transparent coefficients that LR offers, enabling a deeper comprehension of financial relationships. Additionally, LR is computationally effective, making it appropriate for real-time applications and massive financial datasets, both of which are essential in the dynamic world of finance. LR has several drawbacks, though. It is predicated on the idea that variables have linear relationships with one another, which may not always be the case, given how complicated and nonlinear financial markets are. Additionally, its forecast accuracy could be constrained when dealing with complex patterns and high-dimensional data, necessitating more advanced machine learning algorithms for better performance in specific financial modelling scenarios. However, being aware of these benefits and drawbacks enables financial professionals to employ LR in financial research and forecasting jobs confidently.

II.6.2. Lasso Regression

Lasso regression, short for Least Absolute Shrinkage and Selection Operator Regression, has gained prominence in finance for its ability to handle high-dimensional datasets and mitigate the issue of multicollinearity. Lasso regression is a linear regression technique enhanced with L1 regularization, which minimizes the sum of squared residuals, akin to traditional linear regression, but with an added penalty term proportional to the absolute values of the coefficients. The objective function of Lasso Regression can be expressed as Tishbirani (1996):

$$\min_{\beta_0,\beta_1,\ldots,\beta_n} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (Y_i - (\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_n X_{in}))^2 + \lambda \sum_{j=1}^{n} |\beta_j| \right\}, \tag{4}$$

where Y_i represents the dependent variable, the target or outcome variable of interest. The variables $X_{i1}, X_{i2}, \ldots, X_{in}$ collectively denote the independent variables, where X_{i1} signifies the first independent variable, X_{i2} the second, and so on up to X_{in}. These independent variables, or features or predictors, predict the dependent variable. The coefficients $\beta_0, \beta_1, \ldots, \beta_n$ correspond to the coefficients associated with these independent variables, indicating the weights assigned to each independent variable within the linear combination used to predict Y_i. Additionally, λ is employed as the regularization parameter, a pivotal element in Lasso regression, controlling the degree of regularization applied to the model. This parameter significantly influences the trade-off between bias and variance and determines the sparsity of coefficient estimates.

Lasso regression has several benefits for data modelling and analysis. Its main advantage is that it can simultaneously do feature selection and parameter estimation. Lasso efficiently finds and prioritises the most important characteristics by promoting sparsity in the coefficient estimates, improving model interpretability and lowering overfitting. Furthermore, Lasso regression may automatically pick a subset of the most insightful variables, which can assist in avoiding multicollinearity concerns when working with high-dimensional datasets. Lasso does have certain restrictions, though. It has a propensity to randomly choose one variable from a set of strongly correlated predictors, which could result in biased coefficient estimates and miss significant correlations. Furthermore, selecting the regularisation parameter λ can be challenging, requiring careful tuning for optimal model performance. Despite these drawbacks, Lasso regression remains a valuable tool in various fields, balancing interpretability and predictive accuracy.

II.6.3. Ridge Regression

Ridge Regression (RR), a regularised linear regression method, plays a pivotal role in financial modelling as it can address multicollinearity and overfitting. RR extends the conventional linear regression by introducing L2 regularisation, which minimises the sum of squared residuals while adding a penalty term proportional to the square of the coefficients (James et al. (2013)).

The objective function of RR can be expressed as:

$$\min_{\beta_0,\beta_1,\ldots,\beta_n} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (Y_i - (\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_n X_{in}))^2 + \lambda \sum_{j=1}^{n} \beta_j^2 \right\}, \tag{5}$$

where Y_i is the target variable. $X_{i1}, X_{i2}, \ldots, X_{in}$ are predictors. $\beta_0, \beta_1, \ldots, \beta_n$ are their respective coefficients. λ is the regularization parameter governing model complexity.

RR is a modelling approach that has several benefits. It excels at managing multicollinearity and introduces L2 regularisation to increase the stability of coefficient estimations. RR is reliable for datasets with many predictors since this regularisation technique efficiently reduces overfitting. It also provides some degree of robustness against noisy variables, which helps provide more resilient modelling results. RR's disadvantage, however, is that it cannot do variable selection because it keeps all predictors, which may make the model more difficult to interpret.

II.7. Support Vector Machine

Support Vector Machine (SVM) is a powerful supervised learning algorithm commonly used in classification and regression (Bhavsar and Panchal (2012)). It is popular in various domains, from computer science and engineering to finance, healthcare, and natural language processing. In finance, it is used for diverse purposes, including fraud detection, credit risk assessment, and stock price prediction. It also aims to find a hyperplane that best separates data points belonging to different classes or predicts values as accurately as possible.

The objective of SVM, particularly in the case of binary classification, is to find a hyperplane that maximizes the margin between two classes while minimizing classification errors. The following equation defines the decision boundary:

$$w \cdot x + b = 0, \tag{6}$$

Where; w represents the weight vector that defines the hyperplane's orientation, x represents the feature vector of the data point, while b is the bias term. The goal is to find w and b so that the margin between the hyperplane and the nearest data points of each class is maximized. This margin is crucial for the algorithm's ability to generalize well to new data. The decision function for classification is given as follows:

$$f(\mathbf{x}) = \text{sign}(w \cdot \mathbf{x} + b) \tag{7}$$

Here, the SVM aims to find w and b that maximise the margin between the two classes while minimising the classification error.

SVM is a prominent ML technique for tackling complex classification and regression problems. SVM is very useful for datasets with many predictors because it handles high-dimensional feature spaces. Moreover, SVM can identify the optimal hyperplane that maximizes the margin between different classes. This skill enhances its generalization capabilities and reduces the risk of overfitting. In addition, SVM is robust to outliers in the data, as it primarily relies on support vectors near the decision boundary. Notably, SVM can effectively address non-linear relationships by applying kernel functions, which allows it to capture intricate patterns in the data. However, SVM has some drawbacks, such as its computational cost, particularly when dealing with large datasets.

II.8. Random Forest

Random Forest (RF) is a versatile and widely used ensemble supervised machine learning method for classification and regression tasks, renowned for its robustness and ability to handle complex datasets. In an RF, multiple decision trees are constructed, each trained on a bootstrapped subset of the data with random feature selection (Svetnik et al. (2003)). The final prediction is made by aggregating the predictions of all individual trees, often through a majority vote (for classification) or averaging (for regression). This ensemble approach mitigates overfitting, improves model generalization, and can handle high-dimensional data. RF is applied in various domains, including computer vision, natural language processing, biology, and finance (Al-Hashedi and Magalingam (2021)).

An illustration of the formula for a Random Forest prediction is:

$$\hat{Y} = \frac{1}{N}\sum_{i=1}^{N} f_i(X), \tag{8}$$

where $\hat{Y}$ represents the predicted outcome, N is the number of trees in the forest, and $f_i(X)$ is the prediction of the $i-th$ tree in the forest for input features X.

RF is particularly important in finance because it applies to various financial tasks, such as credit scoring to assess the creditworthiness of borrowers, fraud detection to identify unusual patterns in transactions, portfolio optimisation to make investment decisions, and stock price prediction. Its ability to handle large financial datasets, identify relevant features, and make accurate predictions makes it a valuable tool for financial analysts and institutions.

Numerous benefits of RF include feature importance estimates, strong predictive performance, and resistance against overfitting. High-dimensional data can be handled, and complicated relationships can be captured. Additionally, compared to other methods, it just needs minor hyperparameter tuning. There are restrictions to take into account, though. RF models might not be as easy to interpret as simpler models and can be computationally expensive. When model complexity rises, comprehending the underlying decision-making process may not be easy. Additionally, even while it lessens overfitting, it may still be vulnerable to data noise. Careful consideration of the number of trees and other hyperparameters is required for optimal performance.

II.9. Neural Networks

Neural Networks (NNs), also known as artificial neural networks (ANNs), are a class of ML models inspired by the structure and functioning of the human brain. NNs consist of interconnected nodes organized into layers: an input layer, one or more hidden layers, and an output layer. Each connection between nodes has a weight, and each node applies an activation function to the weighted sum of its inputs (Guresen and Kayakutlu (2011)). NNs are used for various ML tasks, including classification, regression, image and speech recognition, natural language processing, and more.

A straightforward feedforward neural network has the following formula, where f stands for the activation function:

$$\hat{Y} = f(W^{(2)} \cdot f(W^{(1)} \cdot X + b^{(1)}) + b^{(2)}), \tag{9}$$

where $\hat{Y}$ symbolizes the predicted outcome, X corresponds to the input data, $W^{(1)}$ and $W^{(2)}$ denote the weight matrices for the hidden and output layers, respectively, and $b^{(1)}$ and $b^{(2)}$ represent the bias vectors associated with the hidden and output layers, respectively.

NNs offer a wide range of important applications in finance. Notably, they are used in algorithmic trading, where they can quickly react to market changes and analyse large amounts of previous data, enabling automated, data-driven trading methods. In risk assessment, where they model and forecast various risk categories, such as market, credit, and operational risk, neural networks are also crucial. They play a vital role in fraud detection by spotting odd trends or abnormalities in user behaviour or transaction patterns and protecting financial institutions from fraud. NNs are also a key component of credit scoring models, offering a more precise assessment of creditworthiness by considering a wider range of factors and patterns in applicants' financial histories.

NNs provide several benefits, such as their capacity to handle high-dimensional data, simulate complicated relationships, and adapt to different tasks. They eliminate the need for intensive feature engineering since they can automatically extract pertinent features from unprocessed data. Deep NNs with many hidden layers can also learn hierarchical representations, enabling them to handle tasks that get more difficult as they get more complicated. However, there are problems with NNs. To avoid overfitting, they frequently require a lot of data, especially deep NNs; they demand a lot of computer power. It can be hard to comprehend why a specific prediction was made since NNs models' interpretability is sometimes constrained. Training NNs can be time-consuming, and hyperparameter tuning can be complicated. Nevertheless, they are an important instrument in the finance sector due to their capacity to handle complex financial issues and enhance decision-making.

We use the following neural networks from one to five hidden layers. The first is an architecture with one hidden layer of 32 neurons (NN1-32). The second has 1 hidden layer of 100 neurons (NN1-100); NN2-32 has 2 hidden layer of 32 and 16 neurons; NN3-32 has three hidden layers of 32, 16, and 8 neurons; NN4-32 has four hidden layers of 32, 16, 8, and 4 neurons; NN5-32 has five hidden layers of 32, 16, 8, 4, and 2 neurons; and finally, NN5-100 has five hidden layer of 100 neurons for each hidden layer.

II.10. Decision Tree

Decision Tree (DT) is a class of supervised ML algorithms employed for classification and regression applications. It is frequently used in many fields, such as engineering, finance, marketing, and healthcare, and they are straightforward but effective decision-making tools.

In a DT, the nodes stand in for features or attributes, the branches for decision rules, and the leaves for results or class labels. The method chooses the feature that, given a set of criteria, provides the best split at each tree node as it is formed through a recursive process (Kotsiantis (2013)). As interpretable models, DT is useful for comprehending and elucidating decision-making. The formula for a simple decision tree can be represented as follows:

For classification:

$$\hat{y} = f(feature\_split). \tag{10}$$

For regression:

$$\hat{y} = average(target\_values\_in\_leaf\_node), \tag{11}$$

where *feature_split* indicates how the data is divided at each decision node based on a specific feature and its threshold, while *target_values_in_leaf_node* represents the actual values or class labels of the target variable associated with the data points in a leaf node, which are used to make predictions.

DT holds significant importance and wide applicability in finance, where it is commonly deployed for tasks such as credit risk assessment, fraud detection, portfolio management, and options pricing. DT assist in evaluating the creditworthiness of borrowers by considering various financial attributes and credit history. Furthermore, DT is crucial in identifying potentially fraudulent transactions by recognizing patterns that deviate from the norm. In portfolio management, it aids in asset allocation and investment decisions. Additionally, DT provides a structured framework for evaluating complex financial derivatives in options pricing.

DT provide several benefits, including the capacity to handle both category and numerical data, ease of use, and interpretability. It can be used for classification and regression applications and is robust to outliers. However, DT has drawbacks, including a propensity to overfit complicated datasets if not carefully pruned. When some classes predominate, it might also produce biassed trees. The proper ratio of depth to complexity must be maintained for best results. Nevertheless, DTs are useful tools in industries like finance, where decision-making and model comprehension rely heavily on simplicity and clarity.

II.11. K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm for classification and regression applications. Fundamentally, KNN generates predictions based on the majority class or mean of the values of its K-nearest feature space data points. It is predicated on the fact that related data points typically fall into the same category or have comparable numerical values. KNN classifies new data points by locating the K-nearest neighbours using a distance metric, such as Euclidean distance, and either determining the mean of these neighbours' labels or values (for regression) or giving the class label by majority vote (Imandoust and Bolandraftar (2013)). KNN is adaptable and useful in many fields, including healthcare for disease prediction, recommendation systems for making product or content suggestions, and anomaly detection for finding anomalies in data.

The formula for KNN can be represented as follows, where N represents the set of K-nearest neighbors:

For classification:

$$\hat{Y} = \arg\max \left(\sum_{i \in N} I(Y_i = y) \right). \tag{12}$$

For regression:

$$\hat{Y} = \frac{1}{K} \sum_{i \in N} Y_i, \tag{13}$$

where $\hat{Y}$ represents the predicted class label, N denotes the set of K-nearest neighbors, Y_i signifies the class label of the i-th nearest neighbor, and y corresponds to the class label being predicted for the given data point. Additionally, I(âŃĚ) is an indicator function that equals 1 if the condition inside is true and 0 otherwise. This equation encapsulates the core mechanism of the KNN classification algorithm, where the predicted class is determined by identifying the class label that appears most frequently among the K-nearest neighbors.

KNN is used in finance for stock price forecasting, fraud detection, and credit scoring. By taking into account historical information from comparable borrowers, it aids in determining a person's creditworthiness. KNN uses established patterns of fraudulent behaviour to compare unexpected transaction patterns to identify fraud. KNN can forecast future stock price movements by examining historical price and volume data.

KNN is a useful tool for short categorization assignments due to its simplicity and convenience. Because it is non-parametric, it can easily handle data with intricate decision boundaries. However, its fundamental drawback is its sensitivity to the number of neighbours (K) and the distance metric. KNN can also be computationally expensive, particularly with big datasets. It performs optimally when data is equally distributed among classes or when class imbalances are adequately managed.

II.12. Gradient Boosting

Gradient Boosting (GB) represents a formidable ensemble machine learning technique that assembles a predictive model by composing a collection of weak learners, typically decision trees (Zhang and Haghani (2015)). This process involves sequential training iterations to rectify errors introduced by preceding models within the ensemble. The amalgamation of predictions generated by these weak learners culminates in forming a robust and highly accurate final model. The foundational principle underpinning GB revolves around the iterative refinement of model parameters to optimize a designated loss function, such as mean squared error in regression or log loss in classification tasks. Its manifold applications encompass diverse domains, including predictive modelling, recommendation systems, and anomaly detection, underscoring its versatility and practicality in real-world problem-solving contexts.

The formula for the gradient boosting update step can be represented as follows:

$$F_m(x) = F_{m-1}(x) + \lambda \cdot h_m(x), \tag{14}$$

where; $F_m(x)$ symbolizes the ensemble model at the m-th iteration, $F_{m-1}(x)$ represents the ensemble model after the $m-1$-t iteration, λ serves as the learning rate, determining the magnitude of each iteration's step, and $h_m(x)$ corresponds to the weak learner introduced in the m-th iteration. These elements are integral components within the framework of GB, orchestrating the iterative refinement of the predictive model's parameters to optimize its performance.

In finance, GB is pivotal in critical endeavours, including credit risk assessment, stock price forecasting, and algorithmic trading. Its proficiency lies in its adeptness at modelling intricate financial data, discerning nonlinear relationships, and enhancing predictive precision. In the domain of credit risk assessment, GB emerges as an indispensable tool, capable of evaluating the creditworthiness of borrowers through a comprehensive analysis of diverse contributing factors. In stock price forecasting, it leverages historical market data to anticipate future price movements, facilitating informed investment decisions. Furthermore, GB is invaluable in algorithmic trading in identifying nuanced trading signals and patterns and formulating data-driven, judicious trading strategies.

GB presents numerous notable advantages, encompassing heightened predictive accuracy, adaptability to diverse data types, and proficiency in handling missing values and outliers. Its comparative resistance to overfitting positions it favourably against certain alternative algorithms, underscoring its robustness and applicability across a broad spectrum of problem domains. Nonetheless, it is pertinent to acknowledge that GB necessitates substantial computational resources and meticulous hyperparameter optimization. Additionally, its performance may be susceptible to the influence of noisy data and outliers, warranting careful consideration and preprocessing. Notwithstanding these challenges, the algorithm's demonstrable prowess and versatility render it an invaluable asset in finance and other arenas where precise predictive modelling is a cornerstone.

II.13. XGBoost

XGBoost, short for eXtreme Gradient Boosting, addresses a prominent inefficiency in this technique. Specifically, it enhances efficiency by optimising the process of evaluating potential splits for branch creation, particularly when dealing with many features, which could result in an extensive number of potential splits. XGBoost mitigates this inefficiency by analysing the feature distribution across all data points within a leaf node and leveraging this insight to narrow the search space for possible feature splits (Sagi and Rokach (2021)).

XGBoost is extensively employed in credit risk assessment, stock price prediction, and algorithmic trading in finance. It excels in modelling intricate financial data, enhancing predictive accuracy, and contributing to improved risk management. XGBoost evaluates borrowers' creditworthiness using a comprehensive set of factors for credit risk assessment. In stock price forecasting, it analyses historical market data to predict future price movements, aiding investment decisions. In algorithmic trading, XGBoost identifies trading signals and patterns, enabling data-driven trading strategies and effective risk management.

XGBoost boasts advantages, including high predictive accuracy, computational speed, and handling of missing data effectively. Its robustness against overfitting, achieved through regularization techniques, allows it to capture intricate data relationships. However, tuning XGBoost's hyperparameters can be computationally intensive, especially with large datasets. Its complex model structure limits interpretability compared to simpler models. Nonetheless, its predictive power and versatility make XGBoost a valuable tool in finance and other domains where accurate predictive modelling is crucial.

III. Execution Time by Difference Machine Learning Models

In this section, we begin our analysis by assessing the predictive performance of different machine learning models. Subsequently, we examine the investment performance, and finally, we investigate the execution time of these machine learning models.

We use a benchmark setting comprising 10 machine learning models that incorporate all 209 characteristics from 2000 to 2022.[2]

We use out-of-sample R^2 and mean squared error as indicators of predictive performance. Figure I depicts the R^2 for diverse machine learning models. The upper graph shows the R^2 in descending order. XGBoost and gradient boosting exhibit the highest out-of-sample R^2, both surpassing the 0.7 threshold. Conversely, random forest, linear regression, ridge regression, and decision tree models yield slightly lower R^2. Lasso regression, support vector machine, and K-nearest neighbors models yield substantially lower R^2, which is around zero.

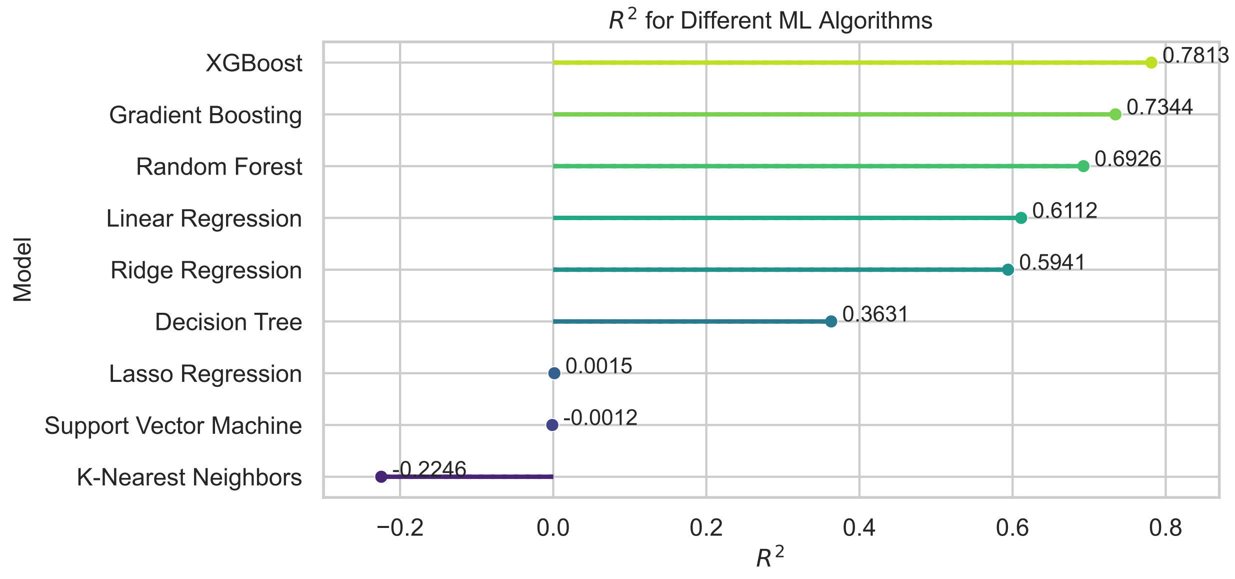
Interestingly, neural networks display substantial negative out-of-sample R^2. The lower graph within Figure I reveals that neural networks with a single hidden layer exhibit the least negative R^2. As the number of hidden layers increases, R^2 ascend, albeit they diminish with an increase in the number of neurons. However, even the most proficient neural network model (NN5-32) still records a substantial negative R^2. This result indicates that our set of 209 characteristics does not facilitate effective stock return prediction using neural networks. Our finding of neural networks is inconsistent with Gu et al. (2020), which find neural networks perform the best.

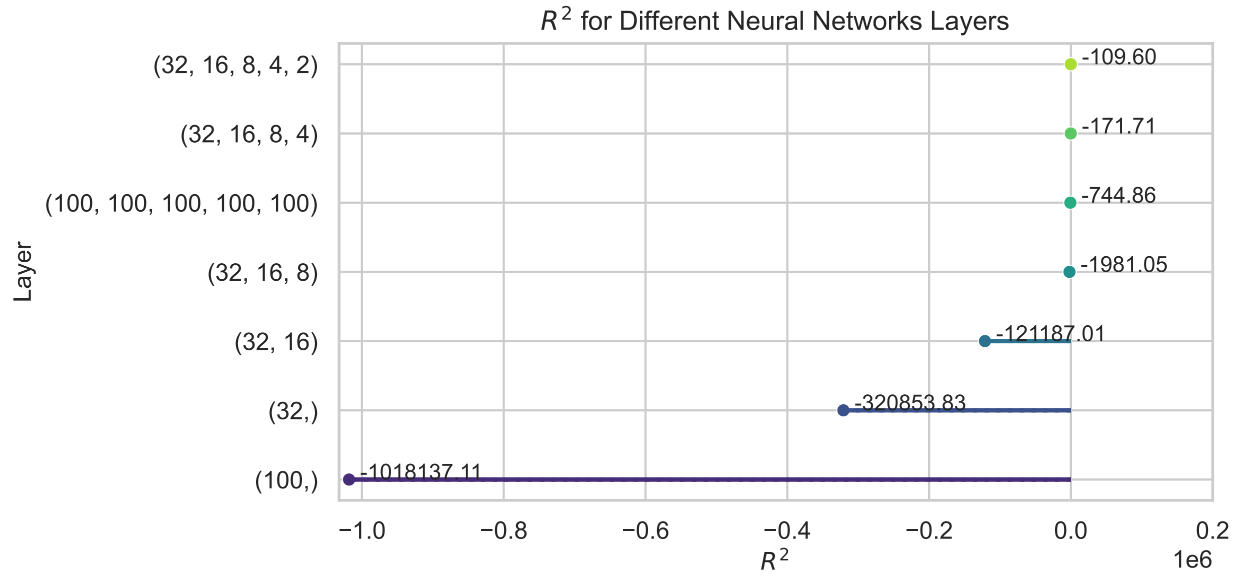
Another measure of the machine learning models' performance is the mean squared error (MSE). It evaluates the average squared difference between the observed and predicted values. Figure II presents the MSE result that is similar to Figure I. Among the models examined, Lasso regression, k-nearest neighbors, and support vector machine exhibit the highest MSE values, exceeding 0.02, indicating comparatively weaker predictive performance. In contrast,

[2]The choice of the 2000 to 2022 time period is arbitrary, yet we will analyze both the entire time frame and distinct sub-periods in forthcoming sections.

Figure I: R-Squared For Different Models

This figure plots the R^2 corresponding to different machine learning models. The upper panel presents data for all machine learning algorithms, excluding neural networks, while the lower panel provides information on neural networks with varying layers and neurons. The sample period is from January 2000 to December 2022.





Random forest, decision tree, gradient boosting, linear regression, and ridge regression display lower MSE values, surpassing 0.02 but remaining below 0.03. Notably, the XGBoost model emerges as the most favorable option, demonstrating the lowest MSE at 0.01, signifying its superior predictive power. Conversely, neural networks yield significantly higher MSE values across various layers, underscoring their limited ability in return predictability. In the subsequent analyses, we exclude neural networks from consideration due to their limited predictive capability.

Up to this point, we have conducted a comparative analysis of various machine learning models using R^2 and MSE as performance metrics. To gain a more comprehensive understanding of the performance of these machine learning algorithms in the context of return predictability, we delve into the investment performance of portfolios constructed by these different machine learning models.

Our approach involves the creation of portfolios that leverage machine learning forecasts to make stock selections. On a monthly basis, each machine learning method is employed to predict stock returns for the subsequent month. These predictions are then used to sort the stocks into deciles, resulting in the formation of ten distinct portfolios, each based on the predicted excess returns generated by a specific machine learning model. These portfolios are equal-weighted and subject to monthly updates.

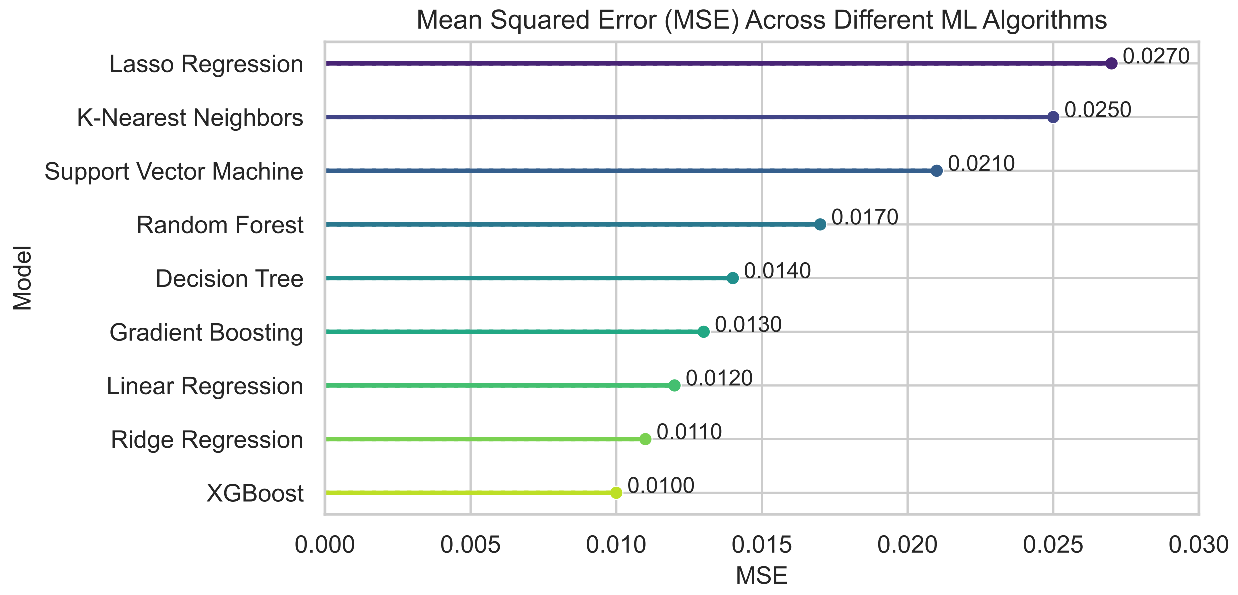
We construct a long-short portfolio strategy, which buys stocks with the highest predicted returns (decile 10) and sells those with the lowest predicted returns (decile 1). This approach allows us to evaluate the investment performance of these machine learning models in a practical context.

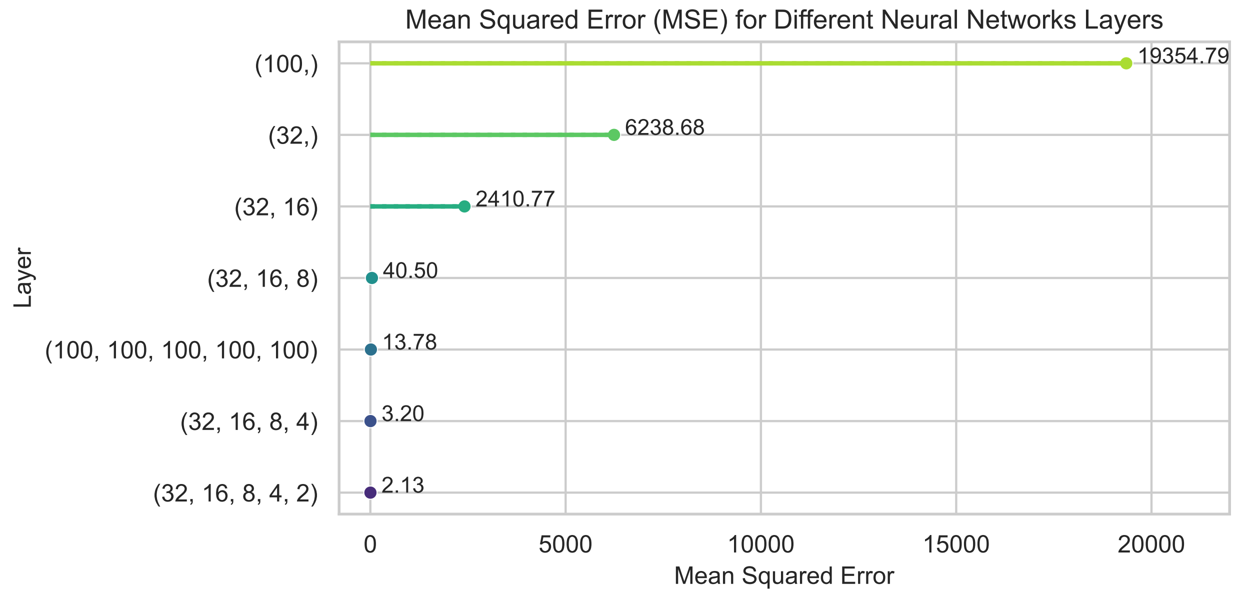
We analyze the Sharpe ratio for the long-short portfolios, which is a widely used metric to measure risk-adjusted returns. Figure III provides an overview of the Sharpe ratios associated with different machine learning models during the period spanning January 2000 to December 2022.

Notably, XGBoost exhibits the highest Sharpe ratio, closely trailed by random forest,

Figure II: Mean Squared Errors For Different Models

This figure plots the Mean Squared Errors corresponding to different machine learning models. The upper panel presents data for all machine learning algorithms, excluding neural networks, while the lower panel provides information on neural networks with varying layers and neurons. The sample period is from January 2000 to December 2022.

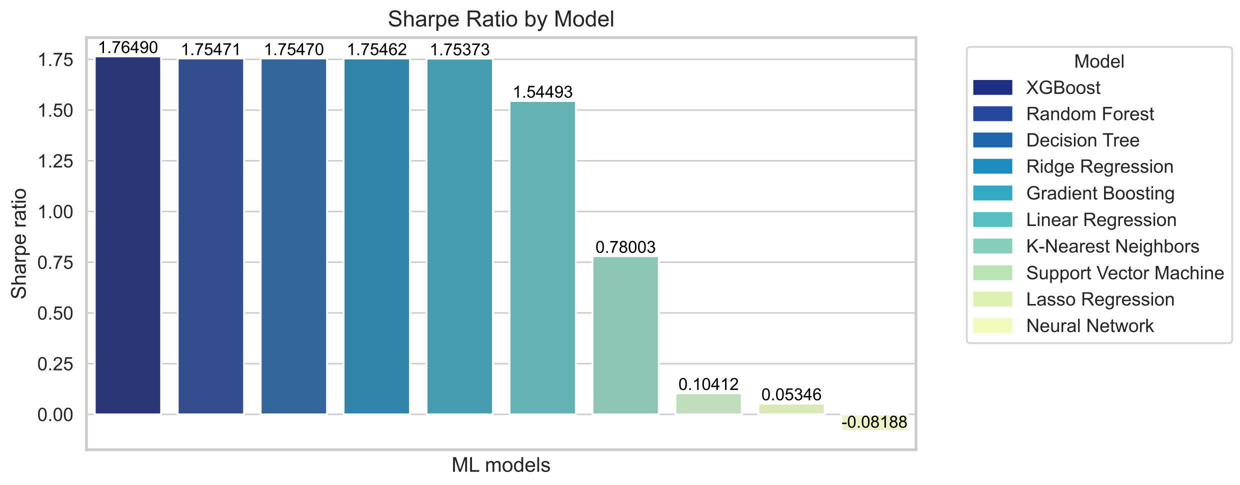




decision tree, ridge regression, and gradient boosting, all of which surpass a Sharpe ratio of 1.75. These models showcase remarkably similar performance in this regard. Linear regression, while still performing well, gives a slightly lower Sharpe ratio of 1.54. K-nearest neighbors, with a Sharpe ratio of 0.78, offer respectable performance but fall short of their counterparts. In contrast, support vector machine, lasso regression, and neural network lag behind, displaying comparatively lower Sharpe ratios. This analysis provides valuable insights into the risk-adjusted returns achieved by each machine learning model.

Figure III: Sharpe Ratio For Different Machine Learning Models

The figure presents the Sharpe ratios of equally-weighted long-short portfolios corresponding to different machine learning models. The sample period is from January 2000 to December 2022.



In conclusion, the investment performance of long-short portfolios constructed using machine learning models aligns with the statistical performance evaluated through R^2 and MSE metrics. Specifically, XGBoost stands out as the top-performing model in terms of statistical measures and also delivers the highest economic contribution to investment performance. Gradient boosting and random forest also exhibit favorable performance in both statistical and economic terms.

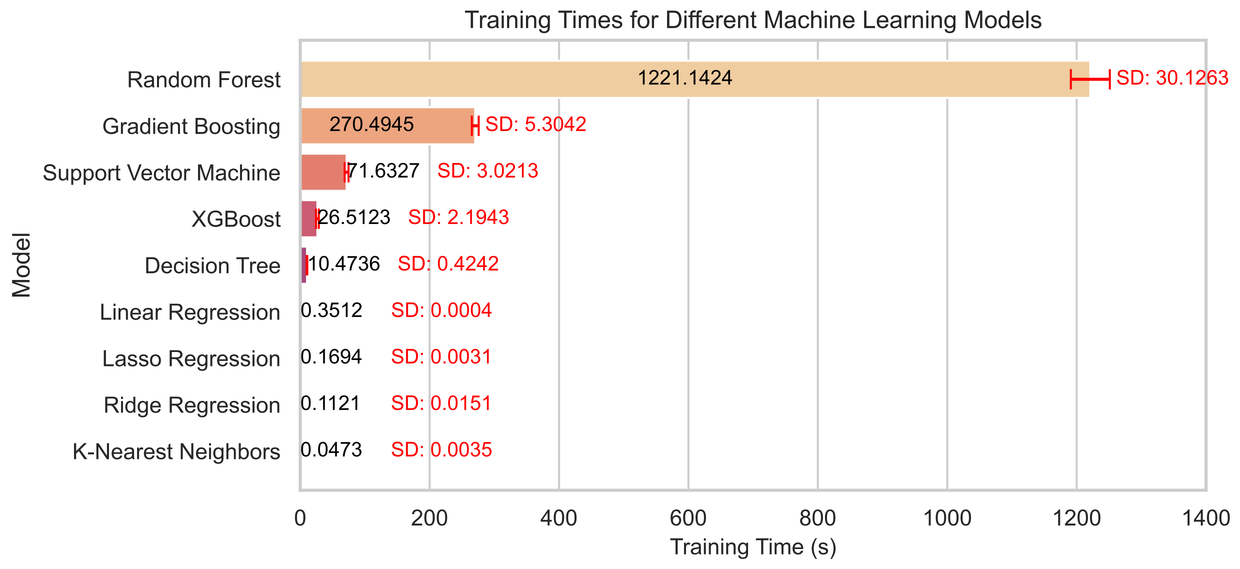
We proceed to examine the execution time required for predicting returns using various machine learning models. Figure IV provides an ascending order depiction of training and prediction times for these models. Linear regression, lasso regression, and ridge regression exhibit very short training and prediction times, both falling below 0.5 seconds. K-nearest neighbors display an exceptionally brief training time but a significantly longer prediction time. Decision tree boasts a reasonable training time along with a short prediction time. XGBoost, the most accurate model, presents a moderately short training time of 26.5123 seconds and a concise prediction time of 0.0332 seconds. In contrast, the second-best model, gradient boosting, demonstrates a considerably longer training time (270.4945 seconds) but a prediction time similar to XGBoost (0.0343 seconds). Lastly, random forest incurs an extremely protracted training time, while support vector machine experiences notably extended prediction times.[3]

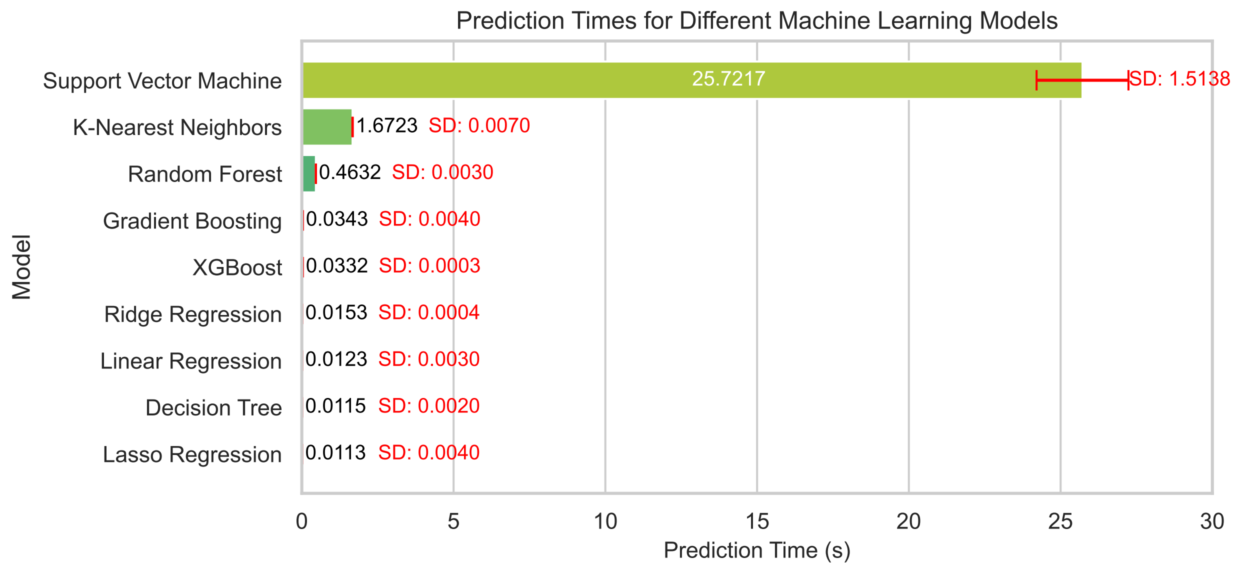
In summary, XGBoost outperforms gradient boosting in terms of both R^2 and overall execution time. While random forest and decision tree exhibit reasonably high R^2, their training and prediction times are suboptimal. The primary trade-off between predictive performance and execution time emerges between XGBoost and regression models (linear and ridge). XGBoost surpasses regressions by at least 0.16 in terms of R^2, yet the training times of regression models are significantly shorter than XGBoost. This positions regression models as viable choices in achieving a balance between accuracy and time efficiency. However, given the considerably lower R^2 and Sharpe ratios exhibited by linear models in comparison to XGBoost, our subsequent analyses concentrate on XGBoost, offering two methods to enhance its time efficiency within the asset pricing framework.

[3]Due to the poor predictive performance of neural networks, we exclude neural networks in this analysis. The training and prediction time for neural networks can be found in the Appendix A.1.

Figure IV:
Training Time And Prediction Time For Different Machine Learning Models

This figure shows the training time and prediction time for different machine learning models excluding neural networks. The sample period is from January 2000 to December 2022.





IV. Reducing The Number of Characteristics

Having established XGBoost as the optimal machine learning model for return predictability, we employ XGBoost for further analyses aimed at enhancing training and prediction efficiency. Various strategies can be employed to optimize machine learning models for time efficiency. One such approach involves the reduction of characteristics to improve algorithmic efficiency.

Figure V illustrates that the majority of characteristics have minimal contribution to equity return prediction. Out of the 209 input variables, only 15 exhibit an importance score exceeding 1. Therefore, it is prudent to consider reducing the number of characteristics as a means to enhance machine learning execution time. We implement this reduction based on specific importance criteria. Initially, we utilize all 209 input variables, subsequently removing characteristics with importance scores below various thresholds, including 0, 0.1-0.2, 0.2-0.3, 0.35, 0.4, and 0.5. This stepwise reduction allows us to investigate the impact of fewer characteristics on both time efficiency and predictive performance.

Figure V: Feature Importance By XGBoost

This figure shows the feature importance by XGBoost. The sample period is from January 2000 to December 2022.

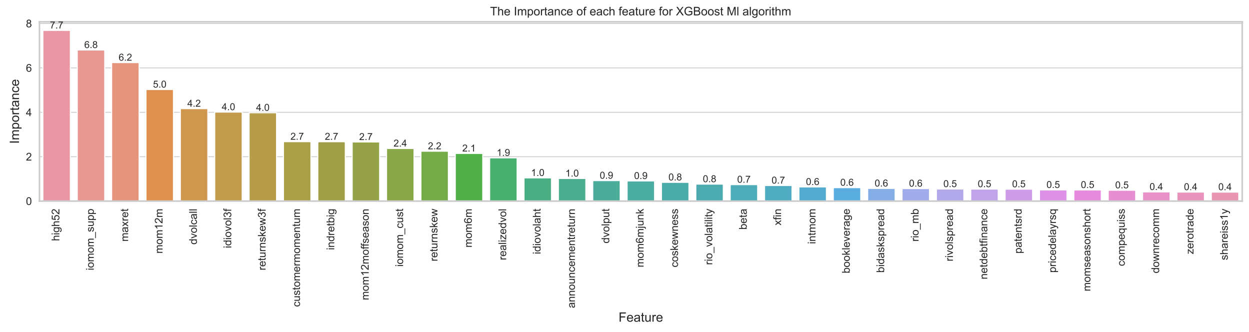


Figure VI displays the training time and prediction time for the top-performing machine learning model, XGBoost, as the number of input variables is varied from those with an importance score exceeding 0.5 to all variables. Training time exhibits an increasing trend with the inclusion of more characteristics. For datasets containing fewer than 57 variables with an importance score greater than 0.26, the training time remains under 10 seconds, signifying a relatively rapid process. When 106 variables with an importance score higher than 0.2 are included, the training time extends to 17.48 seconds. Incorporating variables with an importance score below 0.1 further extends the training time to approximately 25 seconds. The standard deviation of training time also demonstrates growth with the number of characteristics, ranging from 0.75 seconds for variables with an importance score exceeding 0.5 to 3.34 seconds for all variables.

The prediction time illustrated in Figure VI follows a pattern akin to the training time. Prediction times remain below 0.02 seconds for datasets comprising variables with an importance score exceeding 0.24. As more features are incorporated, prediction times range from 0.03 seconds to 0.04 seconds. This aligns with the observation from the benchmark models in Figure IV, where prediction times are notably shorter than training times.

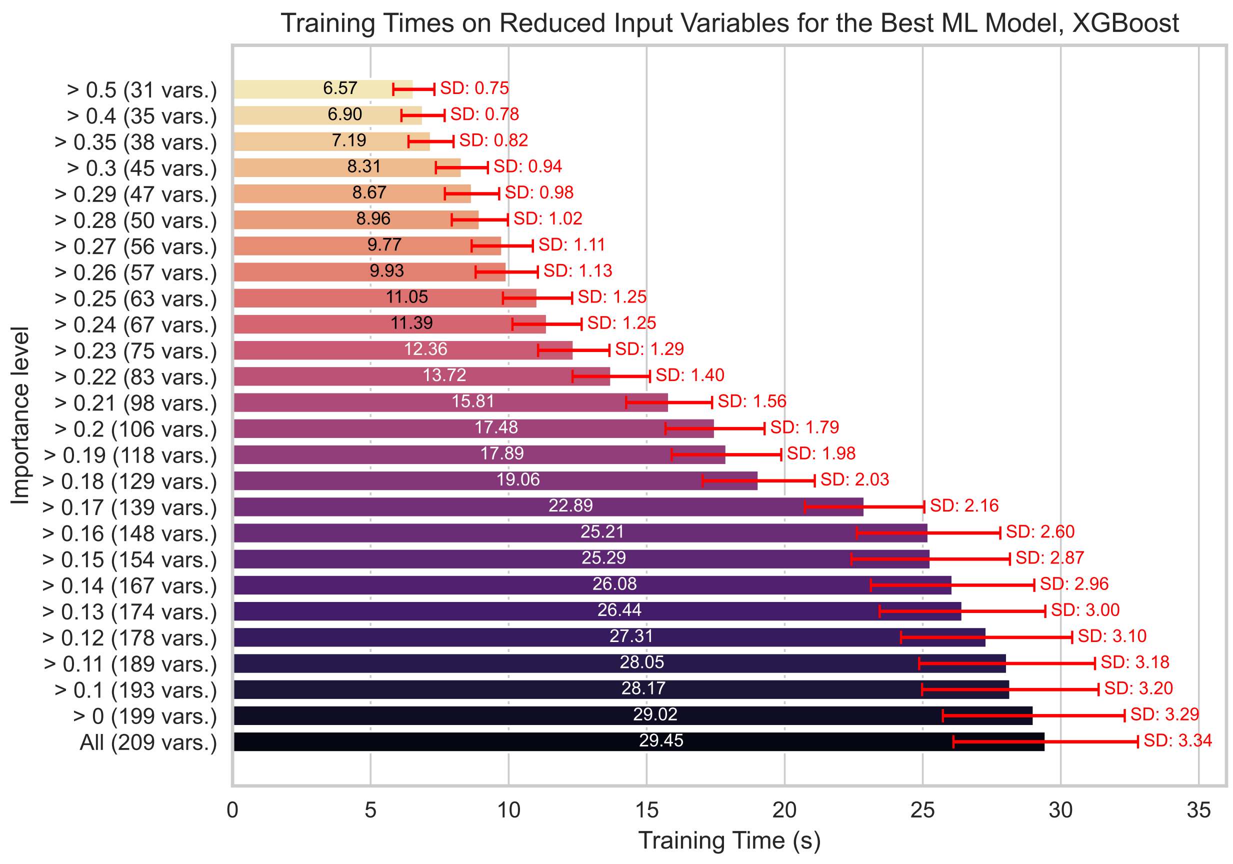
Figure VII shows the out-of-sample R^2 obtained through the reduction of the number of characteristics, guided by their importance levels. To facilitate a more intuitive observation of the slight differences between R^2, we employ a logarithmic scale for data visualization. Notably, as we include only the important characteristics, there is a discernible increase in R^2, albeit of modest magnitude. This observation underscores the beneficial impact of reducing irrelevant characteristics within machine learning models, ultimately enhancing their predictive performance.

We next present the Sharpe ratio achieved by XGBoost while systematically reducing the number of characteristics in Figure VIII. We use a logarithmic scale for data visualization as well. The results show that as less important characteristics are removed, the Sharpe ratios tend to increase slightly. However, the differences among the Sharpe ratios remain minimal. For instance, the Sharpe ratio for the 209 variables is 1.7447, while the Sharpe ratio for the 31 variables is 1.7714, indicating a mere difference of 0.0267. The Sharpe ratio exhibits a pattern akin to that observed in the R^2.

Finally, we use regression analysis to quantitatively assess the impact of saved machine

Figure VI: Execution Time by Reducing The Number of Characteristics

This figure shows the training and prediction times employing the optimal machine learning model, XGBoost, while varying the number of characteristics. The number of characteristics increases with the importance level. The sample period is from January 2000 to December 2022.



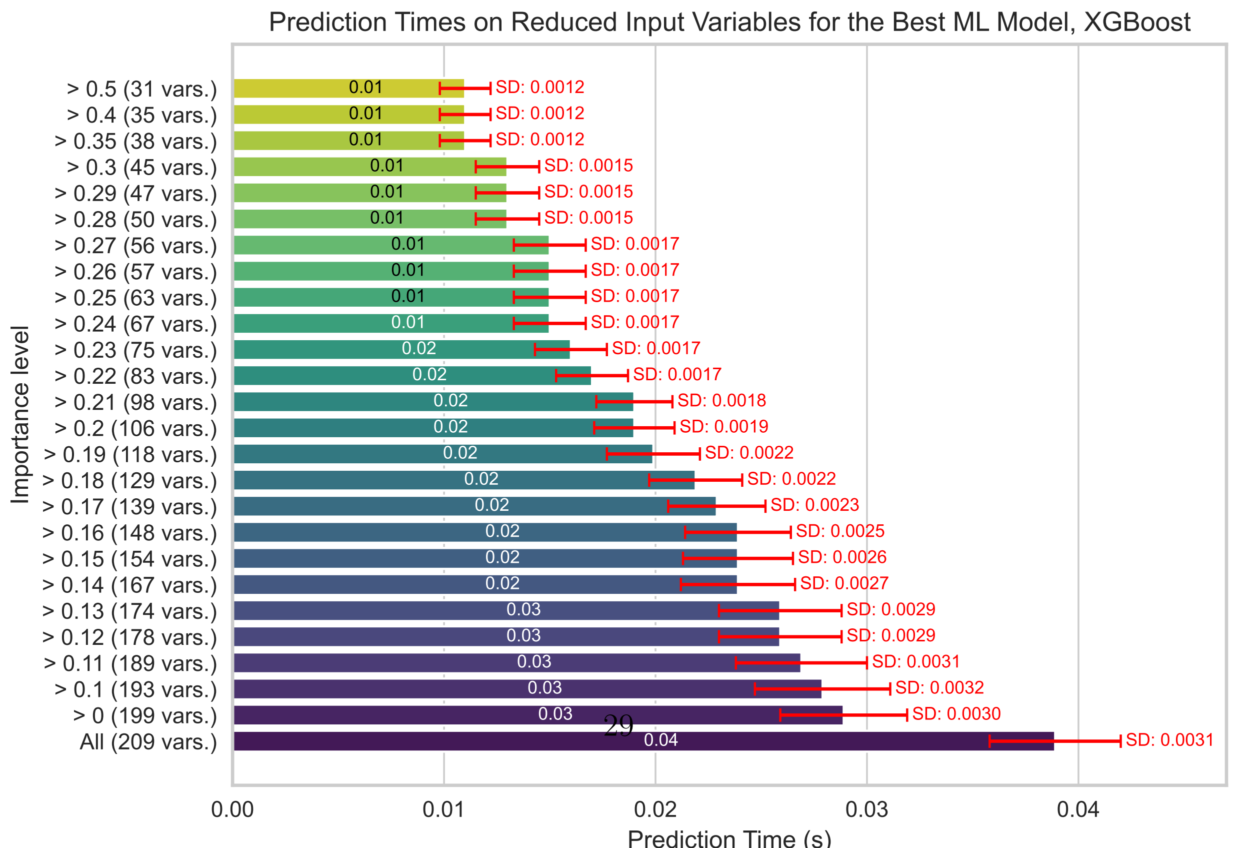


Figure VII: R-Squared by Reducing The Number of Characteristics

This figure shows the R-Squared using the best machine learning model XGBoost. The sample period is from January 2000 to December 2022. The number of characteristics is different.

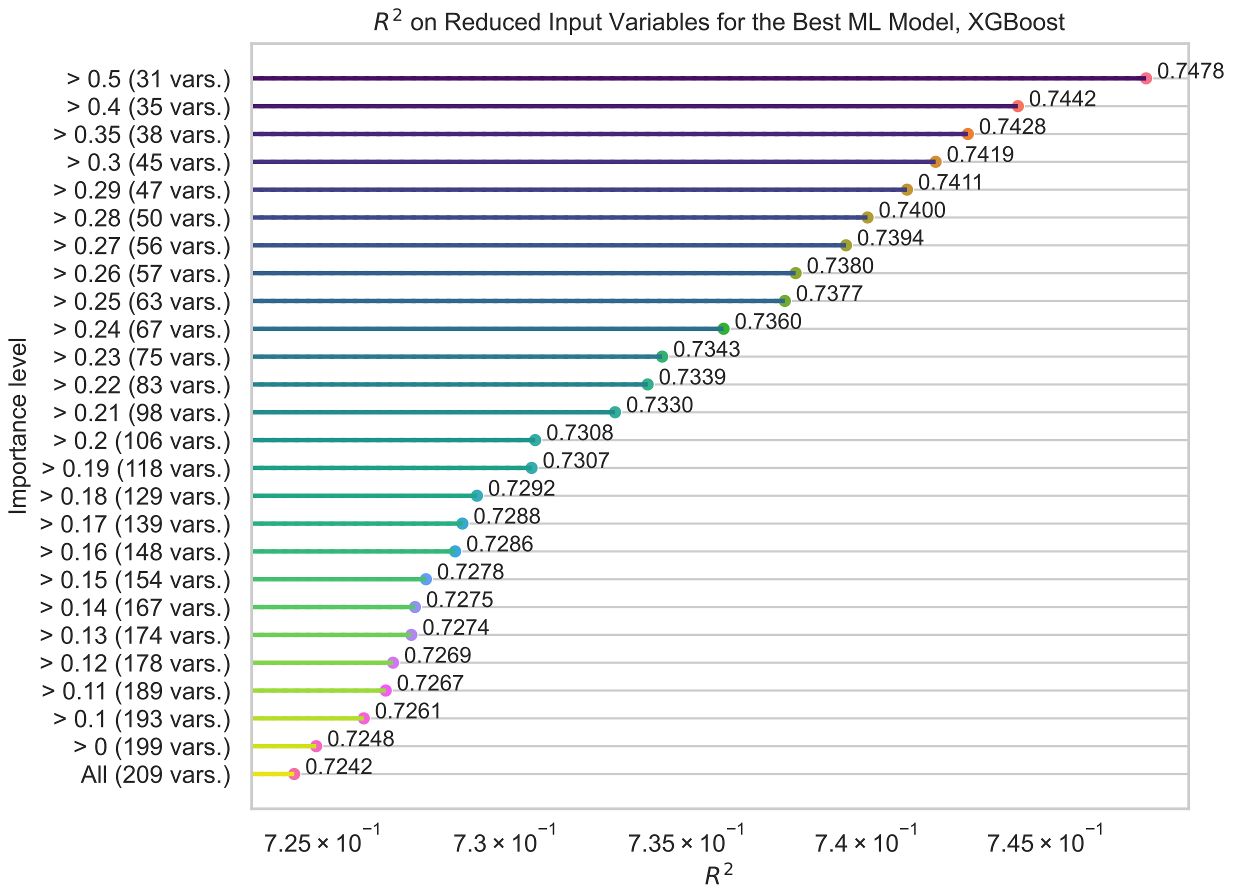
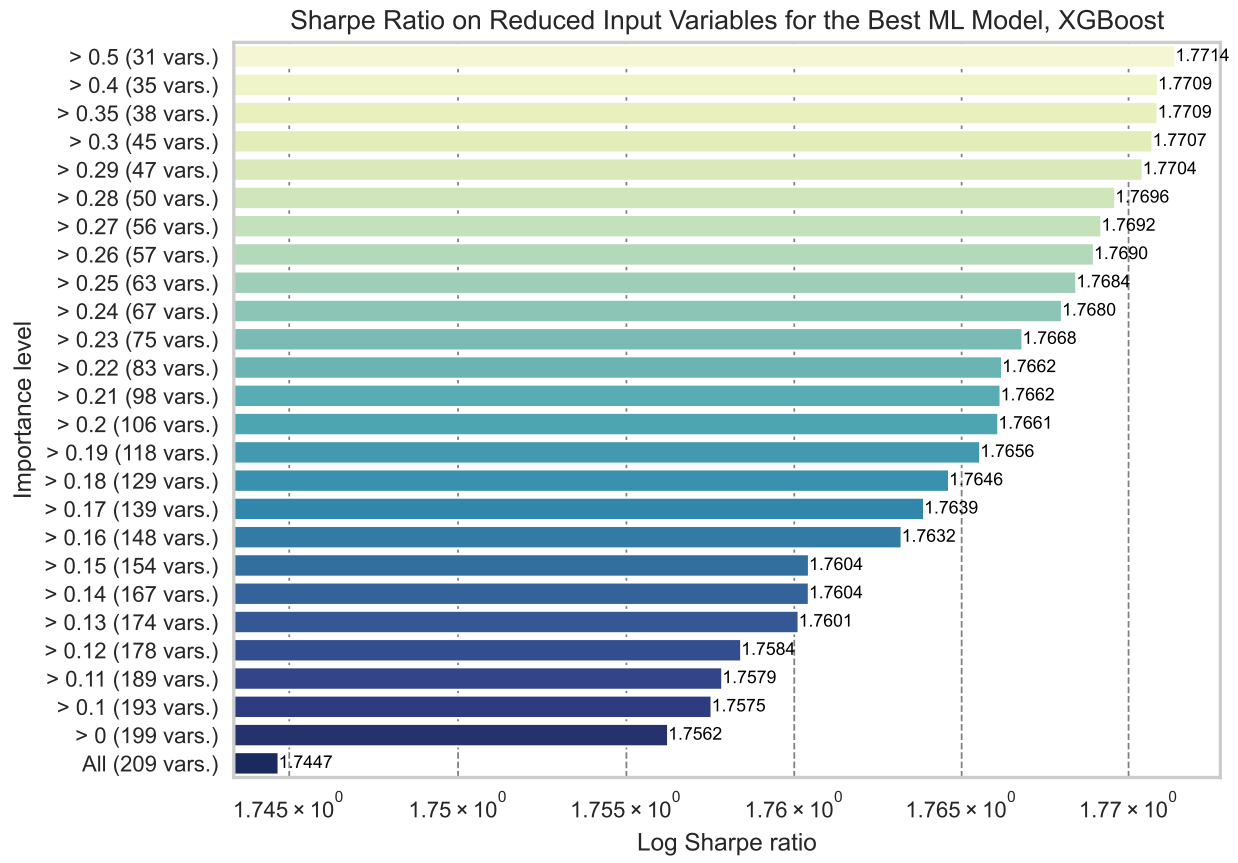


Figure VIII: Sharpe Ratio by Reducing The Number of Characteristics

This figure shows the Sharpe ratios of equally-weighted long-short portfolios using the best machine learning model XGBoost. The sample period is from January 2000 to December 2022. The number of characteristics is different.



learning time on out-of-sample R^2 and the Sharpe ratio. In Table I, our results reveal that by reducing the number of characteristics, a one standard deviation decrease in training time corresponding to a increase of 0.376 in out-of-sample R^2. However, this effect is insignificant for prediction time. A similar pattern emerges for the Sharpe Ratio, where a one standard deviation decrease in training time (prediction time) is associated with a significant increase of 0.878 (0.912) in the Sharpe Ratio. Given that execution time is predominantly comprised of training time, as demonstrated in Figure VI, these training time results align with the overall execution time outcomes. Consequently, our results underscore the dual benefits of feature reduction in enhancing both time efficiency and investment performance, emphasizing the importance of feature shrinking strategies that are well discussed in the literature (e.g., Kelly et al. (2019); Feng et al. (2020); Freyberger et al. (2020)).

Table I: Impacts by Reducing The Number of Characteristics

This table shows the impacts of saved machine learning time on out-of-sample R^2 and the Sharpe ratio by reducing the number of characteristics. The best machine learning model XGBoost is used in this analysis. Sharpe ratio is calculated using equally-weighted long-short portfolios. The sample period is from January 2000 to December 2022. The training time, prediction time, and execution time are standardized. They depend on the different number of characteristics.

	Out-of-sample R^2			Sharpe Ratio		
Training Time	-0.376			-0.878		
	(-2.03)			(-9.15)		
Prediction Time		-0.212			-0.912	
		(-1.09)			(-11.10)	
Execution Time			-0.376			-0.878
			(-2.03)			(-9.15)
Observations	27	27	27	27	27	27
R-squared	0.141	0.045	0.141	0.770	0.831	0.770

V. Reducing The Time Observations

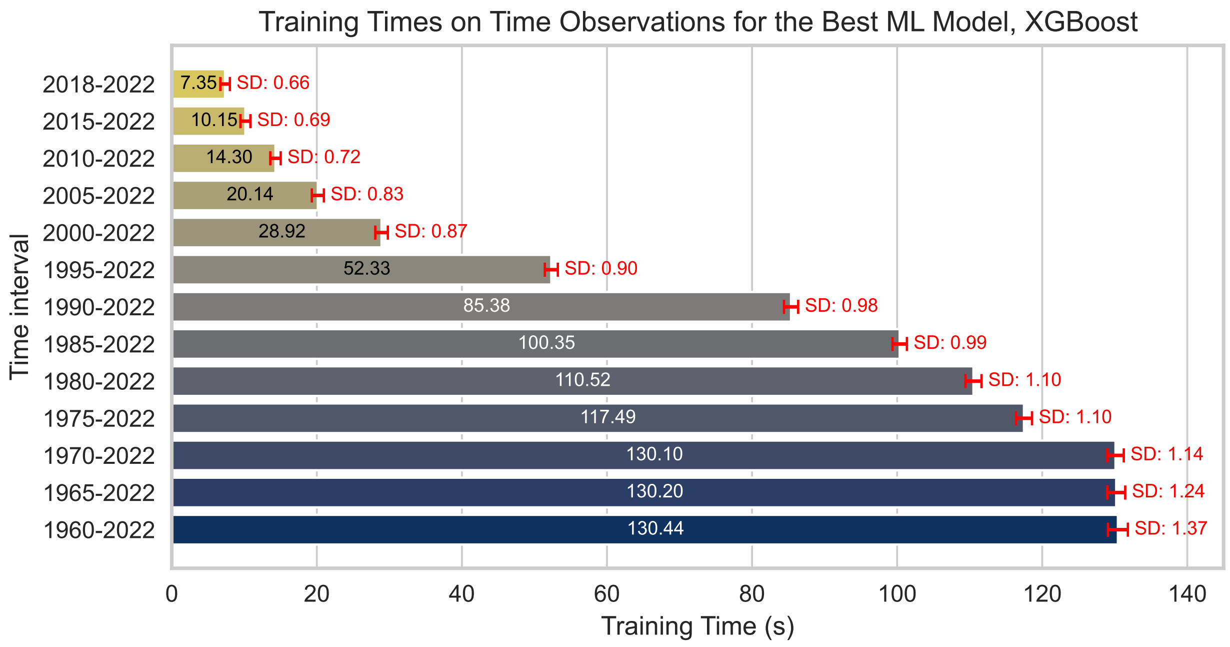
We also explore an alternative method to enhance algorithm efficiency, which involves limiting the number of time observations. Utilizing the optimal machine learning model, XGBoost, and all 209 characteristics, we vary the number of time observations used for training and prediction data. Figure IX shows the training time and prediction time across different time observation periods. It is evident that the training time increases as the number of years included in the dataset grows. Notably, when utilizing data from the most recent 4 years, the training time for XGBoost is a mere 7.35 seconds, with a standard deviation of 0.66 seconds. Subsequently, as the training period extends, the training time exhibits a consistent upward trend, eventually stabilizing at 130 seconds when spanning from 1970 to 2022, encompassing over 50 years of data.

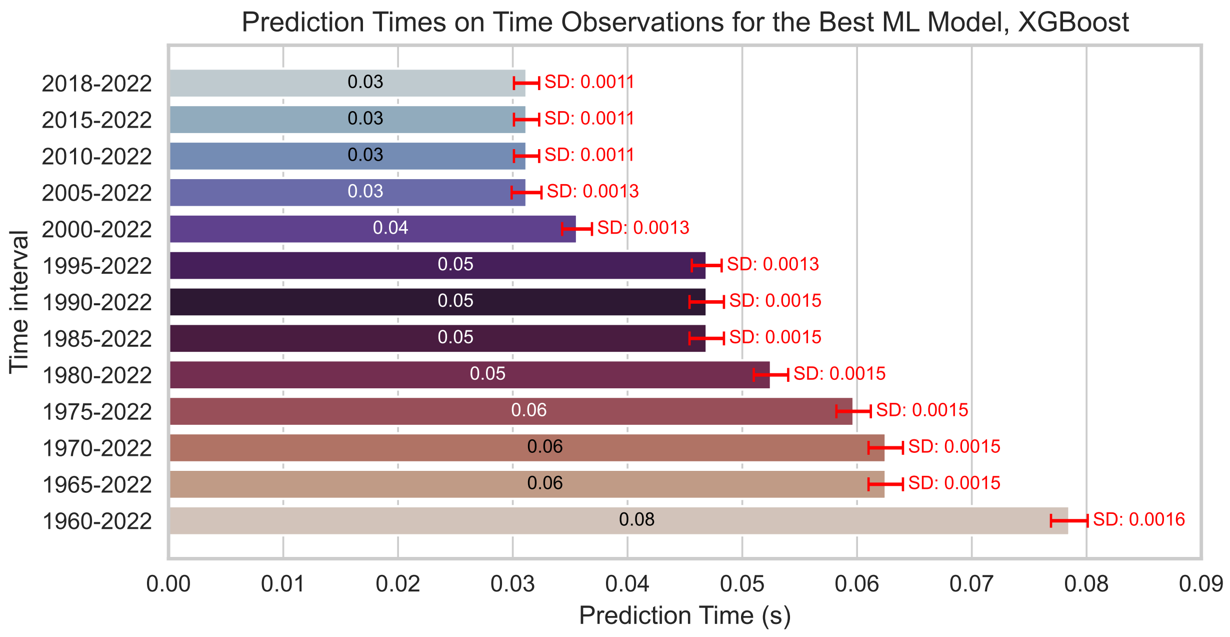
The prediction time also experiences an increase with the extension of the training period, albeit to a lesser extent compared to training times. For instance, the prediction time using XGBoost from 2018 to 2022 is a mere 0.03 seconds, while expanding the training period to cover data from 1960 to 2022 results in a prediction time of 0.08 seconds. In summary, reducing the number of time observations can notably decrease training times while having a less pronounced impact on prediction times.

We finally evaluate the investment performance to gain insights into how altering the time observation periods affects investment outcomes. Figure X shows the Sharpe ratios corresponding to different training and prediction time observations. The graph exhibits a U-shaped pattern. Specifically, the Sharpe ratio is 1.7632 when considering the entire time period, but it declines as the oldest time periods are removed. The Sharpe ratios begin to rise again when the time horizon encompasses the recent three decades (1990-2022). This upward trend continues until the most recent four years (2018-2022), with a Sharpe ratio of 1.8027. Notably, this value surpasses the Sharpe ratio computed using the time span of 1960-2022. This finding suggests that employing machine learning with shorter time periods

Figure IX: Execution Time by Reducing Time Observations

This figure shows the training time and prediction time utilizing the optimal machine learning model XGBoost with all 209 input variables. The sample period is different.

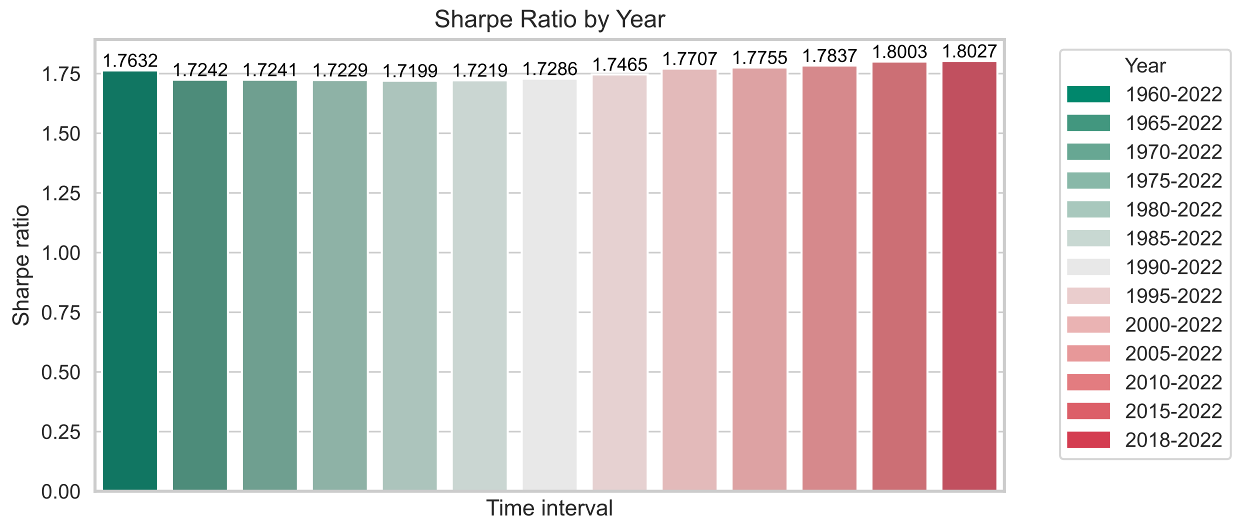




for training and prediction can lead to even higher investment performance. Conversely, a more extensive time period also offers advantages over a median time period in terms of investment performance.

Figure X: Sharpe Ratio by Reducing Time Observations

This figure shows the Sharpe ratios of equally-weighted long-short portfolios constructed using the best machine learning model, XGBoost, with all 209 input variables. The sample period is different.



VI. Conclusion

This paper highlights the critical role of machine learning model execution time in empirical asset pricing. Our study offers a comprehensive evaluation of execution time across various models and introduces two time-saving strategies: feature reduction and a reduction in time observations. Notably, XGBoost emerges as a top-performing model, combining high accuracy with relatively low execution time compared to other nonlinear models. Furthermore, our findings underscore the interplay between model accuracy and execution time in enabling good investment decisions.

For future studies, exploring additional time-efficient algorithms and assessing execution time in different financial contexts could yield valuable insights. Additionally, future research may explore the daily data for forecasting the next day's equity returns. In this context, the temporal efficiency of machine learning algorithms becomes more pivotal, given the necessity for swift investment decision-making based on the predictions generated from daily trading data.

References

Al-Hashedi, K. G. and P. Magalingam (2021). Financial fraud detection applying data mining techniques: A comprehensive review from 2009 to 2019. *Computer Science Review 40*.

Amihud, Y. (2001). Illiquidity and stock returns: cross-section and time-series effect. *Journal of Financial Markets 5*, 1–26.

Ang, A., R. J. Hodrick, Y. Xing, and X. Zhang (2006). The Cross-Section of Volatility. *The Journal of Finance LXI*(1), 259–299.

Bali, T. G., H. Beckmeyer, M. Moerke, and F. Weigert (2021). Option Return Predictability with Machine Learning and Big Data. *SSRN Electronic Journal 36*(202), 3548–3602.

Bhavsar, H. and M. H. Panchal (2012). A Review on Support Vector Machine for Data Classification. *International Journal of Advanced Research in Computer Engineering Technology 1*(10), 2278–1323.

Bianchi, D., M. Büchner, and A. Tamoni (2021). Bond Risk Premiums with Machine Learning. *Review of Financial Studies 34*(2), 1046–1089.

Bryzgalova, S., M. Pelger, and J. Zhu (2019). Forest Through the Trees: Building Cross-Sections of Stock Returns. *SSRN Electronic Journal*.

Chen, A. Y. and T. Zimmermann (2022). Open Source Cross-Sectional Asset Pricing. *Critical Finance Review*.

Daniel, K. and S. Titman (2006). Market reactions to tangible and intangible information. *Journal of Finance 61*(4), 1605–1643.

De Bondt, W. and R. Thaler (1985). Does the Stock Market Overreact? *The Journal of Finance 40*(3), 793–805.

Emmanuel, T., T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, and O. Tabona (2021). *A survey on missing data in machine learning*, Volume 8. Springer International Publishing.

Fama, E. F. and K. R. French (1992). The CrossâĂŘSection of Expected Stock Returns. *The Journal of Finance 47*(2), 427–465.

Fama, E. F. and K. R. French (2008). Dissecting anomalies. *Journal of Finance 63*(4), 1653–1678.

Feng, G., S. Giglio, and D. Xiu (2020). Taming the Factor Zoo: A Test of New Factors. *Journal of Finance 75*(3), 1327–1370.

Freyberger, J., A. Neuhierl, and M. Weber (2020). Dissecting Characteristics Nonparametrically. *The Review of Financial Studies 33*(5), 2326–2377.

Gu, S., B. Kelly, and D. Xiu (2020). Empirical Asset Pricing via Machine Learning. *Review of Financial Studies 33*(5), 2223–2273.

Guresen, E. and G. Kayakutlu (2011). Definition of Artificial Neural Networks with comparison to other networks. *Procedia Computer Science 3*, 426–433.

Hou, K., C. Xue, and L. Zhang (2018). Replicating Anomalies. *The Review of Financial Studies 33*(614), 2019–2133.

Imandoust, S. B. and M. Bolandraftar (2013). Application of K-Nearest Neighbor (KNN) Approach for Predicting Economic Events : Theoretical Background. *Int. Journal of Engineering Research and Applications 3*(5), 605–610.

Israel, R., B. T. Kelly, and T. J. Moskowitz (2020). Can Machines 'Learn' Finance? *SSRN Electronic Journal*, 1–21.

James, G., D. Witten, T. Hastie, and R. Tibshirani (2013). *An Introduction to Statistical Learning*, Volume 112.

Jegadeesh, N. and S. Titman (1993). Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency. *The Journal of Finance 48*(1), 65–91.

Jönsson, P. and C. Wohlin (2004). An evaluation of k-nearest neighbour imputation using lIkert data. *Proceedings - International Software Metrics Symposium*, 108–118.

Kelly, B. T., S. Pruitt, and Y. Su (2019). Characteristics are covariances: A unified model of risk and return. *Journal of Financial Economics 134*(3), 501–524.

Kotsiantis, S. B. (2013). Decision trees: A recent overview. *Artificial Intelligence Review 39*(4), 261–283.

Kozak, S., S. Nagel, and S. Santosh (2020). Shrinking the cross-section. *Journal of Financial Economics 135*(2), 271–292.

Miu, T. and P. Missier (2012). Predicting the execution time of workflow activities based on their input features. *Proceedings - 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, SCC 2012*, 64–72.

Sagi, O. and L. Rokach (2021). Approximating XGBoost with an interpretable decision tree. *Information Sciences 572*, 522–542.

Sanders, R. (1987). The pareto principle: Its use and abuse. *Journal of Consumer Marketing 4*(1), 47–50.

Svetnik, V., A. Liaw, C. Tong, J. Christopher Culberson, R. P. Sheridan, and B. P. Feuston (2003). Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling. *Journal of Chemical Information and Computer Sciences 43*(6), 1947–1958.

Tishbirani, R. (1996). Regression shrinkage and selection via the Lasso.

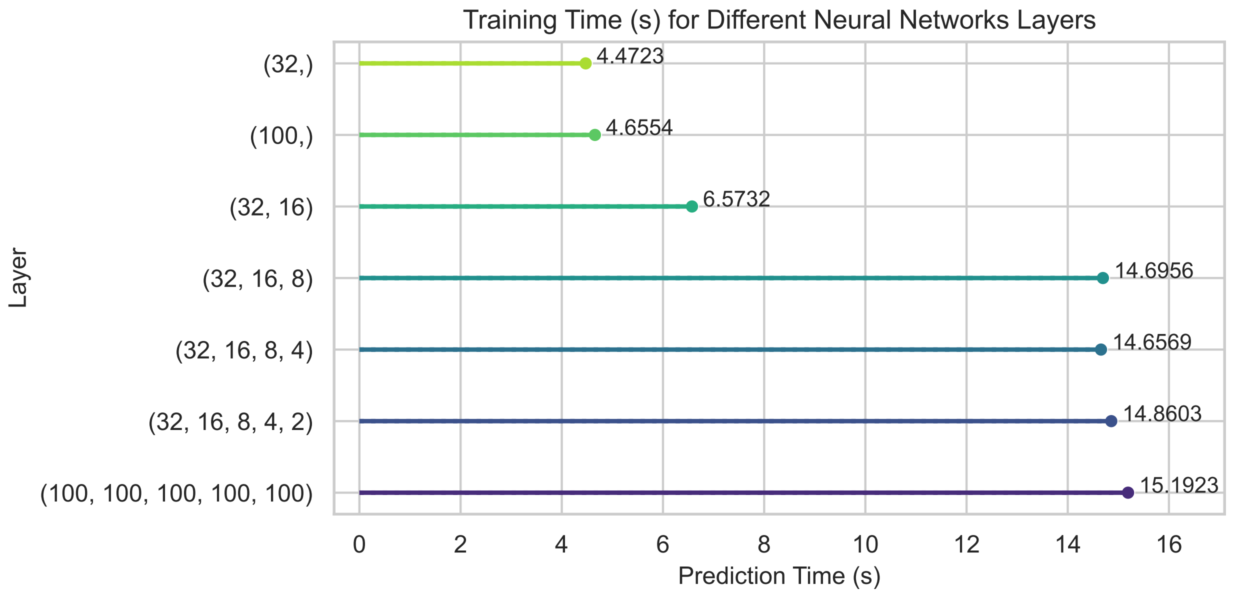
van Binsbergen, J. H., X. Han, and A. Lopez Lira (2020). Man versus Machine Learning: Earnings Expectations and Conditional Biases. *SSRN Electronic Journal*.

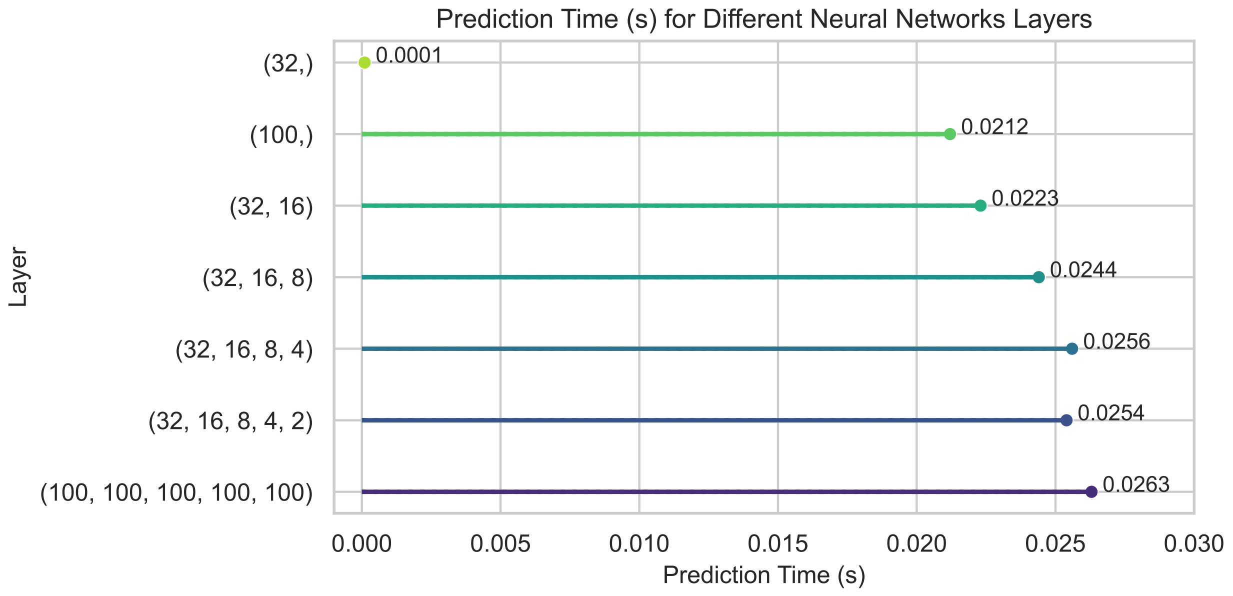
Zhang, Y. and A. Haghani (2015). A gradient boosting method to improve travel time prediction. *Transportation Research Part C: Emerging Technologies 58*, 308–324.

Appendix

Figure A.1: Training Time And Prediction Time For Neural Networks

This figure shows the training time and prediction time for neural networks. The sample period is from January 2000 to December 2022.


Training Time (s) for Different Neural Networks Layers
(32,)
(100,)
(32, 16)
(32, 16, 8)
(32, 16, 8, 4)
(32, 16, 8, 4, 2)
(100, 100, 100, 100, 100)
Layer
Prediction Time (s)


Prediction Time (s) for Different Neural Networks Layers
(32,)
(100,)
(32, 16)
(32, 16, 8)
(32, 16, 8, 4)
(32, 16, 8, 4, 2)
(100, 100, 100, 100, 100)
Layer
Prediction Time (s)