Contents

1 Introduction

In this study, I evaluate the predictive performance of machine learning methods against traditional asset pricing models in forecasting stock returns. In addition, I assess the economic relevance of these predictions by constructing high-minus-low portfolios, to explore whether machine learning methods can be used to profitably extract signals from the market.

The study contributes to the growing literature on machine learning methods for return predictability, that has opened a new strand in the asset pricing literature. With hundreds of factors that have been shown to explain variability in returns, using models that approach high-dimensional data as an opportunity to uncover complex interactions rather than a source of noise and multicollinearity is crucial to enhance predictive performance.

I contribute to the literature by using an extended dataset that runs through the end of 2021. To my knowledge, no other large-scale comparative study has included data this recent. This adds precious information, such as the impactful COVID-19 shock. In addition, I use the existing literature to select the set of tested methods to include the best-performing ones. For this purpose, I also go beyond academic research to include models commonly used in data science competitions. Unlike much of the literature, which often focuses on pre-specified portfolios, I work with data on individual stocks.

I evaluate nine methods, which include OLS, the Fama-French 5-factor model, variable selection methods such as Lasso and Elastic Net, dimensionality reduction techniques such as Principal components regression and Partial least squares, as well as non-linear methods based on gradient boosting (Lgbm and XGBoost) and feed-forward neural networks. I used both a fixed and recursive estimation scheme. The data spans 65 years, from 1957 to 2021, with monthly frequency and over 4 million stock observations with more than a hundred regressors, which include both firm characteristics and macroeconomic variables. For each method, I assess its predictive performance using out-of-sample R^2. The fixed estimation method shows the predominance of variable selection methods, which achieve a R^2_{oos} of 0.34. When switching to the computationally-intensive recursive window, R^2_{oos} generally increases and non-linear methods become the best-performing ones, topped by

neural networks with $R^2_{oos} = 0.49$. The results are coherent and statistically meaningful.

Next, I construct zero-cost, decile-based portfolios with equal and volatility weights, and compute informative portfolio statistics. Machine learning methods lead to significant economic gains. The equal-weighted XGBoost portfolio achieves a Sharpe ratio of 1.19, yielding cumulative returns that are three times higher than those of the S&P 500. The volatility-weighted neural network portfolio achieves a Sharpe ratio of 0.84. These figures significantly improve when constructing centile-based portfolios, to better exploit the large dimensions of the dataset. The XGBoost portfolio achieves a Sharpe ratio of 1.49, and the neural network portfolio of 1.40. Interestingly, the worst-performing portfolios are those with the lowest turnover, suggesting that the performance of machine learning methods comes in large part from frequent portfolio adjustments. Machine learning portfolios seem to work well especially in periods of crisis, during which they outperform the market index the most.

The approach I use focuses on implementability. I conduct the analysis in a way that can be replicated by the average investor in terms of computational constraints. I do not employ powerful computers and explore time/effectiveness trade-offs when tuning parameters and evaluating model performance. The desire to make this study easily approachable also translates in a strong focus on readability. When explaining machine learning methods, I will limit mathematical formalization as much as possible to favor a more intuitive approach that can be better understood and retained by a broader audience. Gaining an initial understanding of machine learning should not be difficult, and excessive formality, while it may be construed as a good signal of intelligence, is absolutely not necessary to implement machine learning methods and obtain meaningful results.

The paper is organized as follows: I begin with a short literature review that traces the evolution of return predictability and the challenges it faces today. Section 2 describes the data used, including the main processing steps, and contains an explanation of the fixed and recursive estimation scheme. Section 3 describes all methods used, with illustrative examples for non-linear methods. Section 4 presents simulation results. Section 5 compares the predictive performance of machine learning methods, as well as that of equal and volatility-weighted portfolios. Section 6 concludes.

1.1 A brief literature review

The question of return predictability has interested researchers for nearly a century. Early studies largely dismissed the notion of predicting stock returns. The first known study to explore this issue is Cowles (1933), which fails to prove that portfolios created by financial experts can outperform the market index. Then, studies such as Fama and Blume (1966) and Jensen and Bennigton (1970) reinforced the case against return predictability. They showed respectively that trading strategies based on the filter rule[1] were not profitable after accounting for transactions costs, and that mutual funds did not manage to consistently outperform the market index.

This branch of literature was instrumental in the rise of the Efficient Market Hypothesis (EMH), formalized by Fama (1970), which claims that stock prices fully reflect available information. According to the EMH, stock prices can be modeled as a random walk and it is thus unfeasible to create profitable trading strategies based solely on historical price or fundamental analysis.

Notwithstanding, the quest of return predictability remained a relevant topic. The literature developed two main approaches: the time-series strand and the cross-sectional strand. The time-series strand conducts predictive regressions where the dependent variable is typically the excess return of a broad market index. The objective of this approach is to forecast future returns using time-varying predictors such as macroeconomic variables or valuation ratios. The most prominent predictor that emerged was the dividend price ratio (Campbell and Shiller, 1988), followed by other macroeconomic variables. These include the earnings-price ratio (Campbell and Shiller, 1988), interest rate spreads (Campbell, 1987), the consumption-wealth ratio (Lettau and Ludvigsson, 2001) and stock volatility (Guo, 2006).[2]

The cross-sectional approach instead focuses on explaining the difference in returns

[1]The filter rule is a trading strategy that attempts to extract trading signals from price movements. For instance, it suggests buying a stock if its price rises by a certain percentage (the "filter") from its most recent low, and viceversa.

[2]Indeed, I also consider the most relevant macroeconomic variables in my analysis, in an attempt to merge the regressors from the cross-sectional and time series strands.

of multiple assets at a given point in time, with the objective of identifying systematic factors that explain part of the variation in returns. These factors are most often asset-specific characteristics. The seminal paper by Fama and MacBeth (1973) provided an econometric tool to robustly assess the significance of potential factors.

Fama-MacBeth regressions consist of two steps. In the initial step, a time series regression is performed. The returns of each asset are regressed on the proposed factor. The estimated coefficient, one for each asset, reflects the asset's exposure to the factor, i.e., it measures the asset's sensitivity to a marginal change in the factor. The second step consists in running a cross-sectional regression in which at each period all asset returns are regressed on the factor exposures. This yields a regression coefficient for each period. These coefficients measure the factor's risk premium, i.e., how much investors are compensated in terms of higher returns for a given exposure to the factor. Finally, the average risk premium is computed, and a standard t-test is employed to assess its statistical significance. This methodology became the standard for nearly all studies that proposed new factors in the cross-sectional strand of asset pricing.

The market factor is the first factor introduced in the literature through the CAPM (Sharpe 1964, Lintner 1965, Mossin 1966), which posits that the excess return of an individual asset is directly related to the excess return of the market. Assets with high market sensitivity, i.e., with a beta larger than 1, are expected to yield even higher returns when the market performs well. The CAPM laid the foundation for the study of risk factors in explaining differences in asset returns. However, researchers started finding anomalies that could not be explained by the CAPM.

Basu (1977) identified the value anomaly, by which value stocks, characterized by a high book-to-market ratio, exhibited higher returns. Banz (1981) discovered the size effect, showing that small-cap stocks earned higher returns than predicted by CAPM. These anomalies revealed the limitations of single-factor models. Fama and French (1992, 1993) addressed this with their famous 3-factor model, which explained a large part of the cross section of returns by using the market, size and value factor. This contributed to the creation of multifactor models, which would soon begin to degenerate.

Carhart (1997) shows that a four-factor model that adds the momentum factor outperforms the three factor model. Fama and French (2015) later extended their framework

to a 5-factor model that adds the profitability and investment factor. In the meantime, the literature had come up with hundreds of factors that claimed to significantly explain the cross section of returns. These factors were tested individually and often exploited anomalies in the data, being the product of data mining.

In recent years, the goal of the literature has shifted towards disentangling the *factor zoo* by assessing the marginal contribution of each factor, in an attempt to reduce the dimensionality of the candidate factors. Machine learning methods are suited for this task, as they can handle high-dimensional data efficiently. Therefore, a third strand of the literature focused on applying machine learning methods to return predictability has recently emerged.

Several studies have applied machine learning methods with promising results, encouraging further research. Feng, Giglio and Xiu (2020) use a double-Lasso procedure to assess the marginal contribution of each factor. Green, Hand and Zhang (2017) identify 12 independent determinants of asset returns. Kelly, Pruitt and Su (2019) employ dimension reduction methods to evaluate factor pricing models. Gu, Kelly and Xiu (2020) and Drobertz and Otto (2021) conduct comparative analyses of machine learning methods in terms of their predictive performance. Giglio, Kelly and Xiu (2022), Bagnara (2024) and Ye et al. (2024) provide insightful literature reviews on the use of machine learning methods in asset pricing. Finally, Rapach and Zhou (2013) reviews the literature on time series and cross-sectional return predictability.

2 Data

The dataset of firm characteristics is based on Gu, Kelly, and Xiu (2020), which itself draws inspiration from Green et al. (2017), and has been updated to include five additional years. It contains monthly observations of 94 firm characteristics[3] from January 1957 to December 2021, covering 65 years. This extended dataset already represents an improvement in the analysis, as it includes a larger sample with an additional macrofinan-

[3]Unfortunately, the majority are updated only on an annual basis, but I will nonetheless perform my analysis with a monthly frequency.

cial shock. In the appendix, I list all the characteristics alongside the papers that found them significant in explaining stock returns. I also provide some descriptive statistics of the covariates by plotting the correlation heatmap (appendix B). Overall, the variables appear not to be highly correlated between each other. Only four pairs of regressors have a correlation that is larger than 0.9 in absolute value.

In addition, I add 8 macroeconomic variables to the set of regressors, which are taken from Welch and Goyal (2008). These have a monthly frequency and have been shown to be drivers of stock returns. Unlike in Gu, Kelly and Xiu (2020), I am unable to make them interact with firm characteristics due to computational constraints. In the simulation section, I will assess the severity of this lack of interaction.

For the dependent variable, I retrieve monthly stock returns for the same period from CRSP and match them to the regressor dataset using the unique *permno* identifier. The dataset includes 32,793 stocks from the NYSE, AMEX, and NASDAQ. Some stocks contain some missing covariates or are not available for the entire span of the dataset. I address the first problem by replacing missing values with the cross-sectional median. Remaining missing values are replaced with the time series median. Finally, if there are still missing values due to the entire row being empty, I replace those values with 0. I keep stocks that are unavailable for the entire length of the dataset to make it more realistic and not engage in survivorship bias. I end up with a dataset with over four million rows (4,096,812) and 102 regressors.

I estimate the model using nine machine learning methods, employing both fixed and recursive windows. When using a fixed estimation scheme, model parameters are estimated only once in the training sample and applied to all future predictions. This implies that data that is generated after the training and validation sample is not used to adjust parameter values. On the one hand, the model is therefore not using new information efficiently and may miss the time-varying component of model parameters. This is particularly detrimental in presence of non-stationarities or structural breaks. On the other hand, fixed estimation bears much lower computational costs, giving the opportunity to estimate the model with a more granular set of hyperparameters in presence of computational constraints.

For the fixed window, the dataset is split into a training sample of 33 years, from

January 1957 to December 1989, a validation sample of 15 years, from January 1990 to December 2004, and a testing sample of 17 years, from January 2005 to December 2021. This tripartition allows for hyperparameter tuning, which greatly influences model performance. In the training sample, each model is estimated using different combinations of hyperparameters. They are then evaluated in the validation sample in terms of predictive performance.[4] The best-performing set of hyperparameters is finally used to conduct out-of-sample forecasting in the testing sample, i.e., the parameters of the model with the chosen set of hyperparameters are estimated in the training sample, stored and used in the testing sample. The latter is truly out of sample as it is neither being used for estimation nor for the tuning of hyperparameters.

For the recurisve window, I start with the same training sample of 33 years, but refit the model each year to incorporate new information. With each iteration, the training sample grows by one year, while the validation sample remains the same size. The testing sample covers only one year at a time. Figure 1 offers a visual representation of the recursive estimation method.

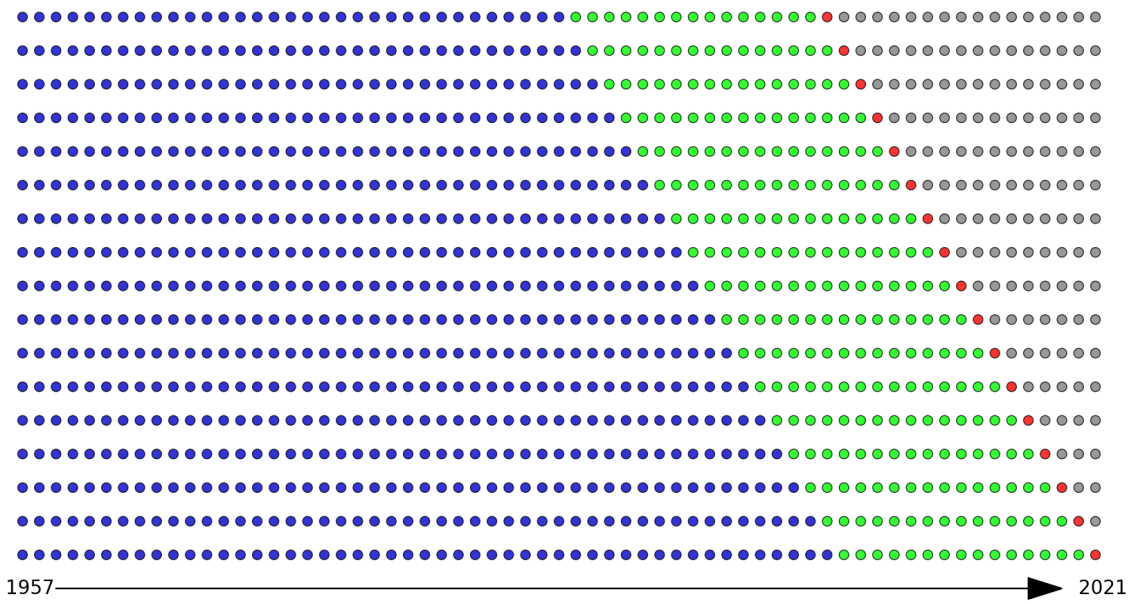

Figure 1: Recursive estimation scheme.

***Note**: Training sample in blue, validation sample in green, testing sample in red. Each dot represents a year.*

[4]The metrics used for such evaluation are described in the simulation section.

3 Methods

In this section, I describe all the machine learning methods applied in this paper. Firstly, let's consider a broad specification of excess returns as being equal to expected returns plus an error term:

$$r_{i,t+1} = \mathbb{E}_t(r_{i,t+1}) + \epsilon_{i,t+1}$$

Each method aims to model expected returns as $\mathbb{E}_t(r_{i,t+1}) = f^*(x_{i,t})$ where $x_{i,t}$ is the set of available regressors and f^* need not to be a function of all the regressors, nor does it have to be linear, as it will depend on the chosen method. The objective of each method is to maximize the out-of-sample explanatory power of the implied model. Depending on the method, this can be achieved by conducting a linear regression, by perfoming variable selection, by creating latent factors or by trying to capture non-linear interactions.

I will first describe the linear methods, i.e., those that define f^* as a linear function of the regressors. Then, I will move on to progressively more complex methods, that offer a more flexible functional form capable of capturing complex interactions between the regressors.

3.1 Ordinary least squares

This is the simplest method, which is often used as a benchmark to show how it can be outperformed by more sophisticated methods.
Using OLS implies that the f^* function is linearly approximated in the following way:

$$\mathbb{E}_t(r_{i,t+1}) = f^*(x_{it}) \approx f(x_{it};\beta) = x'_{it}\beta$$

with x_{it} being the set of regressors and β the corresponding vector of unknown parameters. The pooled OLS estimator is obtained by minimizing an objective function based on the residual sum of squares (RSS), adjusted for the cross-sectional and longitudinal dimensions of the panel dataset. In practice, pooled OLS neglects firm heterogeneity by assuming that all observations belong to the same company.

$$\mathcal{L}(\beta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}(r_{i,t+1} - x'_{it}\beta)^2$$

Lewellen (2015) shows that restricted versions of OLS strongly improve predictive performance and can act as a good benchmark for more sophisticated methods. Therefore, I include an OLS model with deterministically chosen regressors. The candidates are the well-known Fama-French 3-factor model (Fama and French, 1992), Carhart (1997) four-factor model, and the more recent Fama-French 5-factor model (2015), which addresses limitations of the original model, such as its failure to predict the low returns of small stocks, by adding two additional factors.

The original 3-factor model includes the traditional market risk factor, representing the (excess) returns of the market portfolio over the risk-free rate. Fama and French (1992) demonstrates that adding two more factors, value and size, improves the explanation of the cross-section of returns. The value factor captures the outperformance of value stocks (those with a high book-to-market ratio) compared to growth stocks (those with a low book-to-market ratio). The size factor reflects the tendency for small stocks to outperform larger ones, calculated as the excess returns of small stocks over large stocks.[5]

I adjust the original model specification to my setting with individual stocks instead of portfolios, by reducing the set of covariates in the OLS specification. I only include the market factor and the idiosyncratic book-to-market and size factors.

$$\mathbb{E}_t(r_{i,t+1}) = f^*(x_{it}) \approx f(x_{it};\beta) = \beta_1 \cdot mkt_t + \beta_2 \cdot bm_{it} + \beta_3 \cdot size_{it}$$

The persistence of the value and size factors has then been questioned, with data showing that over the past 30 years the value factor is no longer statistically significant (Fama and French, 2020). Over the years new factor models have emerged, with the trend of slightly increasing the number of factors. Inter alia, Carhart (1997) adds the momentum factor to the original FF3 model. It is based on the idea that stocks that have performed well in the recent past, which can range from one month to three years, tend to continue performing well in the near future.

Furthermore, Fama and French (2015) expand their model to include five factors, adding the profitability factor, which measures the excess performance of highly profitable

[5]The authors construct zero-cost portfolios based on the factors used, thus buying value stocks and selling growth stocks, and buying small stocks and selling large stocks.

firms relative to less profitable ones, and the investment factor, which captures the return differences between conservative firms, i.e., with low asset growth, and aggressive firms, with high asset growth.

I select the FF5 model to represent the class of "deterministically restricted" OLS models, as it is the most recent one and tries to build on the shortcomings of the previous versions.

3.2 Lasso and Elastic Net

When dealing with a large number of predictors, OLS becomes a highly inefficient method. Indeed, it ends up overfitting the information contained in the dataset, struggling to distinguish the signal from the noise. This implies that OLS will display a relatively large coefficient of determination in the training sample, but will perform poorly out of sample. I will check whether this expectation, along with many others, is corroborated by both simulated and real data.

The most intuitive remedy is to include a penalty term in the objective function to inhibit the inclusion of unnecessary regressors. This approach is particularly useful in a setting with over 100 predictors, which can all allegedly help explaining the cross-section of returns. However, some factors may be redundant, either due to data snooping or high correlation with other factors, making their marginal contribution negligible.

A common regularization technique is the Lasso method, whose penalty function can be expressed as

$$\phi(\beta; \lambda) = \lambda \sum_{j=1}^{K} |\beta_j|$$

where K is the number of covariates and $\lambda \in [0, 1]$ modulates shrinkage strength. When $\lambda = 0$, the model reduces to OLS, while increasing λ leads to a more parsimonious model. λ is the first hyperparameter encountered. Its optimal value is determined in the validation sample. I estimate the model in the training sample using a reasonable set of parameter values, selecting the λ that maximizes out-of-sample performance in the validation set. This general approach will be used for all models with hyperparameters. When there are more than one, I will test all possible combinations.

The objective function minimized by Lasso is thus the same as for OLS, with the

addition of a penalty function

$$\mathcal{L}(\beta;\lambda) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}(r_{i,t+1} - x_{it}'\beta)^2 + \lambda\sum_{j=1}^{K}|\beta_j|$$

Lasso can be considered a proper variable selection method. The fact that the penalty function uses the L1 norm entails that the coefficients of redundant factors will be brought to zero.[6] Thus, the Lasso model will contain fewer regressors than the OLS model, mitigating the overfitting problem. By deteriorating in-sample performance in favor of a better out-of-sample performance, this method can hopefully constitute an improvement over the OLS benchmark.

I will also implement the Elastic Net penalty function, which can be viewed as a linear combination of the Lasso and Ridge penalty functions. It has been shown to outperform Lasso in many instances, as in Zou and Hastie (2005).

$$\phi(\beta;\lambda,\rho) = \lambda(1-\rho)\sum_{j=1}^{K}|\beta_j| + \lambda\rho\sum_{j=1}^{K}\beta_j^2.$$

To sufficiently differentiate it from Lasso, I will select only values of ρ, that sets the weights of the Lasso and Ridge component, that lie in a sufficiently close interval around 0.5.

Plain Lasso and Elastic Net still remain relatively simple regularization methods for asset pricing. However, some studies have proficiently used more advanced versions, using firm characterstics as regressors. Freyberger et al. (2020) use adaptive group Lasso to select characteristics, coming up with a model that achieves a higher out-of-sample explanatory power than linear panel regressions. Feng, Giglio, Xiu (2020) use double selection Lasso to show that most recently found factors give an insignificant marginal contribution to explaining stock returns. Kozak et al. (2020) use Elastic Net to construct a stochastic discount factor that summarizes the set of firm characteristics.

[6] On the other hand, the other popular penalized linear method, Ridge, contains the L2 norm and is therefore just a shrinkage method, as it will reduce the magnitude of the β associated with redundant regressors, but will never bring it exactly to zero.

3.3 Principal components regression

The main threat posed by penalized linear methods in this analysis is the risk of sub-optimal forecasts. The set of predictors includes many financial ratios and accounting variables, which may exhibit strong comovement by construction. In a setting with highly correlated regressors, variable selection methods may perform poorly (Drobertz and Otto, 2021). Therefore, instead of performing variable selection, it may be more effective to apply dimensionality reduction techniques to the dataset by summarizing information through artificial factors.

In this class, one of the most common methods is Principal Components Regression (PCR). It is comprised of two main steps. First, principal components analysis (PCA) is performed on the set of regressors to condense the predictors into a smaller set of factors. These factors are then be used in the second step as predictors to forecast stock returns. Principal components are linear combinations of regressors that are orthogonal to each other and ranked by their ability to explain the dataset's variance. The process begins with standardizing the data, followed by computing the covariance matrix of the regressors. Next, eigenvectors and eigenvalues[7] are calculated: the eigenvectors represent the principal components. Since the covariance matrix is symmetric, its eigenvectors will all be orthogonal to each other, thus satisfying the definition of principal component. The eigenvalues associated to each principal component (eigenvector) instead represent the amount of variance of the original dataset that is explained by that principal component. The researcher then has to select the k components with the highest eigenvalues so as to balance the trade-off between reducing the dimensionality of the original dataset while still being able to summarize it efficiently. The reduced dataset should still explain a significant portion of the original dataset's variance.

The literature has come up with a plethora of criteria and tests to determine the optimal number of principal components. Bai and Ng (2002) propose several information criteria that differ in how they penalize model complexity. They show these can

[7]For a matrix, an eigenvector is a non-zero vector that, when multiplied by the matrix, only changes in magnitude and not in direction. The eigenvalue is the factor by which the eigenvector is scaled during this multiplication.

consistently estimate the number of factors as the cross-sectional and time dimension diverge. Watson and Amengual (2007) extend this framework to dynamic factors. Heuristic approaches are also frequently used, such as the cumulative explained variance, which consists in choosing the number of principal components that explains a certain percentage of the variance of the original data, typically 70, 80 or 90%.

To remain consistent with the other methods, I use the tripartition of the dataset to determine the optimal number of principal components, selecting the number that provides the best out-of-sample performance in the validation set. Once PCA is complete, the selected components are used as covariates in a simple linear regression.

One of the most natural objections to the use of PCR is that it only considers the covariance among the predictors, neglecting the relationship between the regressors and the target variable. This can result in selected principal components that contribute little or nothing to forecasting the target variable. The next method tries to address this potential shortcoming.

3.4 Partial Least Squares

The Partial Least Squares methodology (PLS) shares many similarities with PCR, as it also involves a transformation step followed by a linear regression. The main difference lies in the first step. In PLS, components are not ranked based on their ability to explain the variance of the original predictor set, but rather on their covariance with the target variable. This implies that PLS will outperform PCR in a scenario in which some of the components that do not explain a lot of the original variance and are hence not selected in PCA exhibit a strong predictive association with the target variable, which is instead captured by PLS.

The difference between the two methodologies can be made clearer by examining their weight selection objective functions. PCR selects weights to maximize the variance of the regressors conditional on all components being orthogonal. In contrast, PLS maximizes the squared covariance between the target variable (stock returns) and the original set of regressors under the same orthogonality constraint.

PLS is a special case of the more general three-pass regression filter (3PRF) introduced

by Kelly and Pruitt (2013),[8] which also is a widely popular forecasting method. I favor the use of PLS due to its easier implementation and lower computational requirements.

Formally, PLS and PCR amount to reducing the dimensionality of the original vectorized model $R = X\beta + \mathcal{E}$ into

$$R = (XW_p)\beta_p + \tilde{\mathcal{E}}$$

where X is an $NT \times K$ matrix, W is the K x P matrix of weights with $P < K$ being the number of chosen components, and β_p is the Px1 vector that stores the OLS coefficients of the selected factors.

A major pitfall of dimensionality-reduction techniques is that the resulting models are made up of artificial predictors which are linear combinations of hundreds of real economic variables and thus have a much more ambiguous interpretation. Retrieving factor loadings can help mitigating the problem, but in the case of a rather uniform distribution of weights it is impossible to give an economic intuition to the chosen components nor have any insights in what are the drivers of expected returns. This lack of interpretability is a recurrent theme with machine learning methods and will be further exacerbated when dealing with more sophisticated non-linear methods.

3.5 Non-linear methods

All the methods discussed so far assume linear relationships and do not account for interactions between predictors. This may be a severe limitation when the phenomenon under study and thus its underlying model is highly complex. In this section I will discuss some of the most used methods that offer greater flexibility at the cost of a higher overfitting risk. Given the vast number of such methods, I will focus only on those that have already been shown to perform well.[9]

[8]The authors of the paper claim that PLS and the 3PRF coincide when *(i) the predictors are demeaned and variance standardized in a preliminary step, (ii) the first two regression passes are run without constant terms and (iii) proxies are automatically select*

[9]For this reason, I will omit inter alia the generalized linear method and random forest, which are consistently beaten by the methods I will describe. See for example Gu, Kelly and Xiu (2020).

3.5.1 Gradient boosting

Gradient Boosting Regression Tree (GBRT) is an ensemble machine learning technique for regression tasks that builds a model in a recursive manner by combining a series of shallow decision trees. Firstly, I will illustrate what a decision/regression tree is.

Regression trees are a type of decision tree whose outputs, commonly termed leaves, represent numerical values. The dataset is divided into subsets based on specific feature thresholds, which are chosen to minimize the residual sum of squares (RSS). For instance, the root node of the tree is determined by the feature and threshold that create the most informative split, i.e., where the partition yields the lowest RSS when predicting average returns in each subset. The process continues by further splitting the data at intermediate nodes according to other feature thresholds. At the end, the dataset is partitioned in groups of observations that display similar characteristics. The predicted return for each observation is the average return in its respective group. To avoid overfitting, the splitting process is generally halted when further splits would result in partitions with too few observations.

Figure 2 provides a simple illustration of a regression tree. It groups all observations in four categories based on the similarity of certain features. With this method, any new observation can be immediately assigned to one of the categories, with its forecasted returns equal to the average return of that group.

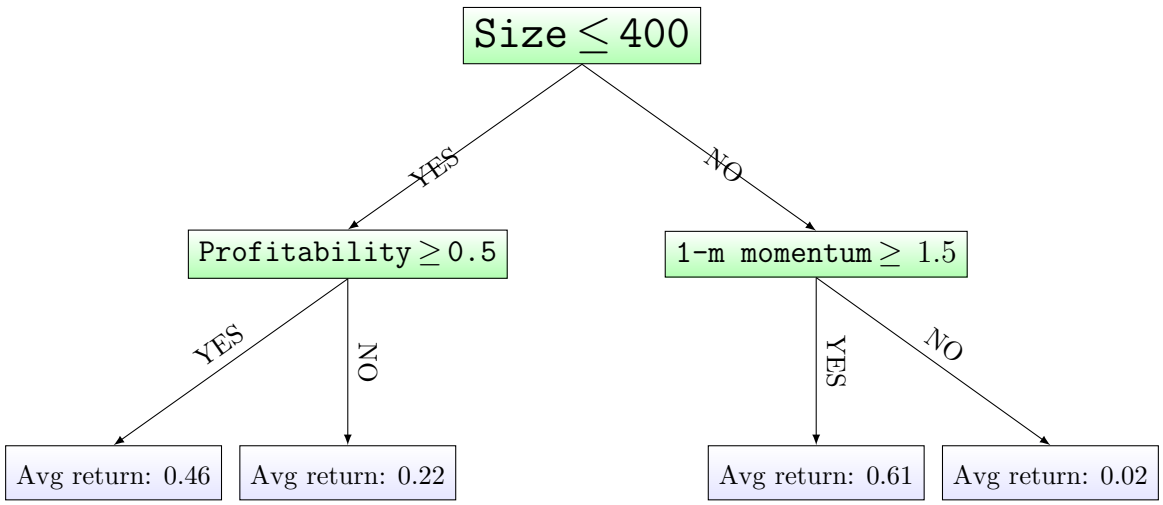

Figure 2: An example of a regression tree.

More formally, consider a generic node m of the tree of the tree with a partitioned dataset D_m. For every feature-threshold combination (j, k), the dataset will be split into two subsets:

$$D_m^l(j,k) = \{(X,y)|x_j \leq k\}$$

$$D_m^r(j,k) = D_m \setminus D_m^l(j,k)$$

where X is a row vector that contains all J features and l, r stands for the left and right splits, respectively. All possible splits are then evaluated and the chosen one will be the minimizer of the mean squared error (MSE) or RSS defined as: [10]

$$\bar{y}_m^l = \frac{1}{n_m^l} \sum_{y \in D_m^l} y \qquad \bar{y}_m^r = \frac{1}{n_m^r} \sum_{y \in D_m^r} y$$

$$MSE(D_m) = \frac{n_m^l}{n_m^l + n_m^r} \sum_{y \in D_m^l} (y - \bar{y}_m^l)^2 + \frac{n_m^r}{n_m^l + n_m^r} \sum_{y \in D_m^r} (y - \bar{y}_m^r)^2$$

Gradient boosting uses regression trees iteratively to generate with high-quality predictions of the target variable. I try to give an intuitive explanation of its functioning, while the detailed algorithm is outlined in the appendix.

An initial guess equal to the average of all returns is made.[11] Then, for each observation, pseudo-residuals are computed as the difference between actual and predicted returns. A regression tree is constructed to predict the pseudo residuals, rather than the returns themselves. The tree is restricted in depth, limiting the number of leaves as a consequence. Since this number is by far smaller than the number of observations, several observations and their residuals will fall under the same leaf. Each leaf is adjusted to represent the average of these pseudo-residuals. Next, each observation is run down the tree and "matched" to its new pseudo-residual. A new prediction can then be made by adjusting the initial guess, the average, by the new pseudo residual scaled by a shrinkage factor, or learning rate. The process is repeated iteratively: new pseudo residuals are computed and a new regression tree is fitted on them. The iterations continue until either the maximum number of trees is reached or no further improvement is made in fitting the

[10] To simplify notation I have omitted (j, k), which should be added to every partitioned dataset.

[11] This is a simplification that stems from considering the following loss function: $L = \frac{1}{2}(y_i - \bar{y})^2$. In this case, the initial guess coincides with the average.

residuals.

The main idea behind gradient boosting is to make many small adjustments to the initial prediction rather than a few large ones, as the former strategy leads to more accurate forecasts. This explains the presence of a learning rate or shrinkage factor, whose role is precisely that of "slowing down" the convergence of the pseudo residuals to zero.

Rossi (2018) uses boosted regression trees to obtain stock returns forecasts that outperform those of benchmark models.

There exist several frameworks to apply gradient boosting regression trees. I will use Light Gradient Boosting Machine (LGBM) and XGBoost, which are unarguably the two most popular.

The main advantage of LGBM is its computational speed and low memory consumption, while still delivering accurate predictions and having high training accuracy. Some studies, such as Łoś et al. (2021) and Yang[12] et al. (2021) have even shown that LGBM outperforms XGBoost in terms of efficacy and predictive performance.

XGBoost, short for Extreme Gradient Boosting, also provides an optimized implementation of gradient boosting. An important feature of XGBoost is the presence of a regularization term in its objective function, which helps to prevent overly complex models, e.g., by removing unnecessary tree branches and to mitigate the risk of overfitting, at the cost of slightly longer computational times compared to LGBM. Since its inception in Chen and Guestrin (2016), XGBoost has rapidly become one of the most used methods for forecasting stock returns in Kaggle competitions, given its consistent superior performance to the majority of other methods. In academia, its adoption has been slower, with Zhang (2022) being one of the few studies that highlights XGBoost's effectiveness.

3.5.2 Neural networks

Neural networks are computational models that are trained to find relationships within the data through complex interactions that should mimic the functioning of the human brain. The incredible flexibility of this method comes at the cost of a lack of interpretability.

[12]While this paper shows that LGBM performs better than XGBoost, it actually shows that an ensemble model that is based on both methods is the best performing one.

Neural networks are often referred to as a black box, as it is nearly impossible to trace and explain how the output is derived. In this section, I will provide an intuition of how neural networks work and include an illustration of a neural network fit to my analysis. In the appendix, I describe the regularization techniques employed.

Neural networks consist of different nodes and connections between them. The nodes of the input layer represent the data, which is typically standardized. The output layer contains the prediction of returns made by the neural network. Between these two layers lie one or more hidden layers, with a certain number of nodes, which transform and add the data to improve the prediction of expected returns. Neural networks are highly parametrized, with two key parameters: weights, which scale the data, and biases, which act as an intercept.

Another crucial element of a neural network is the non-linear activation function, which is applied at each node of the hidden layers and is reshaped by the data and parameters. I will use one of the most common activation functions, the Rectified Linear Unit (ReLU)[13], which is simply defined as:

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise.} \end{cases}$$

Figure 3 gives an illustration of a neural network with an input layer of 3 nodes, a single hidden layer of 2 nodes and an output layer of 1 node. In my application, the input layer will have at least 94 nodes, and I will experiment with neural networks of different depth, i.e., with a number of hidden layers ranging from 1 to 5. Note that Gu, Kelly and Xiu (2020) actually found that a neural network with 3 hidden layers, thus relatively shallow, outperforms deeper networks with 5 layers. That is to say that a deeper network is not necessarily a better performing one.

[13]Other common activation functions include Softmax, sigmoid and hyperbolic tangent. There are also variations of ReLU, such as the exponential linear unit and the Leaky ReLU.

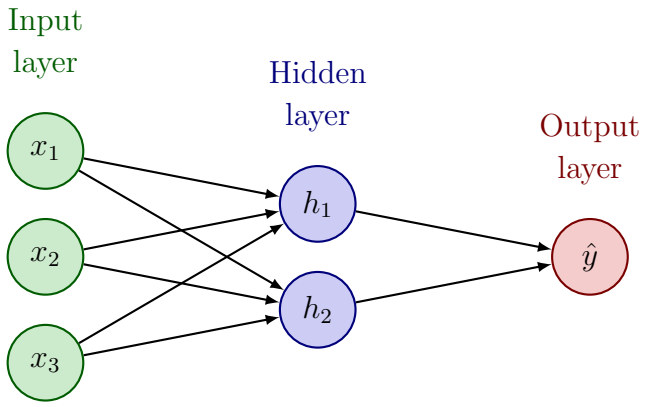

Figure 3: An illustration of a neural network.

We start with a dataset with three features x_1, x_2, x_3. Before reaching a node of the hidden layer, each feature is transformed by an idiosyncratic weight w_{ij}, with i referring to the node and j to the features. The scaled features are then summed and a bias b_i is added. For node 1 of the hidden layer, the signal received is:

$$z_1 = w_{1,1} \cdot x_1 + w_{1,2} \cdot x_2 + w_{1,3} \cdot x_3 + b_1$$

In the hidden layer node, the activation function processes the signal z_1 as input value and produces the output h_1. Thus, $h_1 = f(z_1)$, with f being the activation function. Note that since I am using the ReLU activation function, it will either be that $h_1 = z_1$ when $z_1 > 0$ or $h_1 = 0$.

Then, all the outputs of the hidden layers are scaled by another set of weights w_i, summed and adjusted by a bias b. The output value is given by:

$$\hat{y} = w_1 \cdot h_1 + w_2 \cdot h_2 + b$$

This basic example already has 11 unknown parameters (8 weights and 3 biases) that have to be estimated. Parameters are simultaneously estimated using stochastic gradient descent (SGD), which speeds up the gradient descent process by using a subset of the dataset for each update, reducing computational time.

Gradient descent is an optimization algorithm that uses gradients to find the optimal parameter values. Initial guesses are made: biases are usually set to 0 and weights are randomly drawn from a standard normal distribution. The gradient, i.e., the collection of all partial derivatives of the loss function (RSS) with respect to each parameter, is then

computed. Next, the gradient is adjusted by a learning rate $l \in (0, 1)$. The result of this adjustment determines the step size, i.e., how much the initial guesses will be adjusted in the following iteration. This process continues until the gradient is sufficiently close to the zero vector, or until the maximum number of iterations is reached. Formally, the step size at iteration n is given by:

$$\text{stepsize}_n = l \cdot g_n = l \cdot \begin{bmatrix} \frac{\partial \text{RSS}}{\partial w_{11}(n)} & \frac{\partial \text{RSS}}{\partial w_{12}(n)} & \cdots & \frac{\partial \text{RSS}}{\partial b(n)} \end{bmatrix},$$

$$w_{11}^{(n+1)} = w_{11}^{(n)} - \text{stepsize}_n[1]$$

where l is the learning rate, and g_n represents the gradient of the residual sum of squares with respect to the parameters at iteration n. $w_{11}^{(n+1)}$ is the new guess for this parameter, obtained by subtracting the step size from the previous guess. This process is repeated for each parameter to update all of them in every iteration.

What remains to be explained is how the gradient is computed. An algorithm known as backpropagation is used. It computes the gradients through an efficient use of the chain rule. It decomposes the partial derivative of RSS with respect to a parameter into the product of a sequence of partial derivatives that are easier to compute. For example, the derivative of w_{11} is computed as:

$$\frac{\partial \text{RSS}}{\partial w_{11}} = \frac{\partial \text{RSS}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial h_1} \times \frac{\partial h_1}{\partial z_1} \times \frac{\partial z_1}{\partial w_{11}}$$

Indeed, the link between RSS and the parameter passes through the predicted value $\hat{y}$, the output of the activation function h_1 and the signal z_1.

Figure 4 illustrates a more structured neural network, that resembles the one I use in this analysis. Regarding the choice of the number of nodes in the hidden layers, which can be somewhat arbitrary, I follow the geometric pyramid rule of Masters (1993), which halves the number of nodes in each hidden layer. Based on Gu, Kelly and Xiu (2020), the first layer has 32 nodes, the second 16, the third 8, and so on.

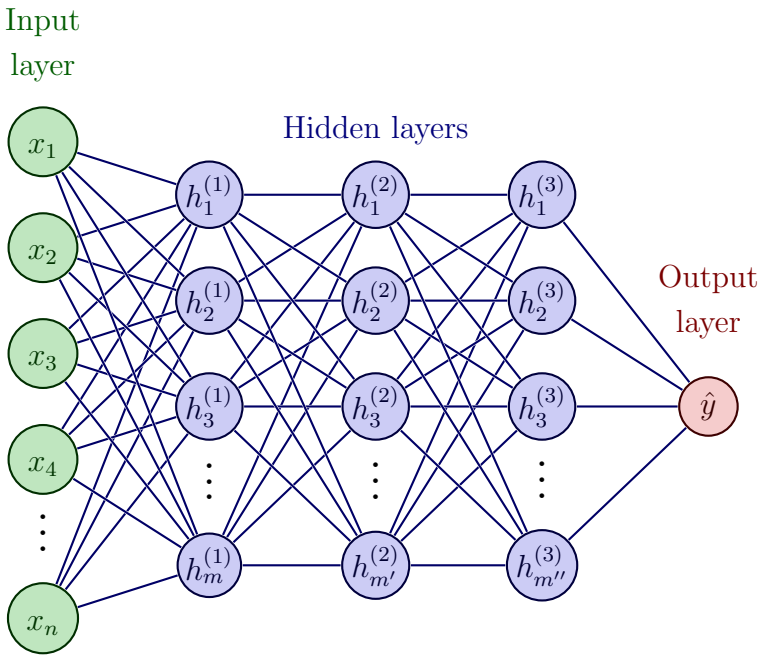

Figure 4: An example of a deep neural network.

Eldan and Shamir (2016) discuss the interesting dilemma of whether to prioritize a wider (more nodes in each hidden layer) but shallower neural network or a deeper and narrower one, when building a network with bounded size. They maintain that the latter structure is to be preferred. Therefore, in the empirical part of this analysis, I will compare the predictive performance of a 3-layer network with 32, 16 and 8 nodes respectively, against that of a deeper 4-layer network with 16, 8, 4 and 2 nodes.

Finally, in the appendix, I will describe some regularization methods that are frequently used in the literature and which I also employ, such as batch normalization, early stopping, learning rate shrinkage and ensembles.

Neural networks have been widely used in the literature for stock return prediction. Chen, Pelger and Zhu (2019) use various types of neural networks and show they can produce the stochastic discount factor with the highest Sharpe ratio. Moghar and Hamiche (2020) uses a recurrent neural network to predict stock prices for specific stocks, yielding promising results. Gu, Kelly and Xiu (2020) find that a neural network with 3 layers is the best performing model for predicting stock returns. Qiu, Song and Akagi (2016) successfully use neural networks to make predictions on Japanese stock prices.

4 Simulation models

Before starting to work with real data, I conduct experiments with simulated data to gain initial insights into the performance of different machine learning models. This requires making assumptions on the data-generating process which can hopefully approximate real data. A major assumption is the number of latent factors in the *true* model. I assume a true 5-factor model, which can be summarized by the following set of equations:

$$\begin{cases} r_{it+1} = f^*(x_{it}) + e_{it+1} \\ e_{it+1} = \beta_{it} v_{t+1} + \epsilon_{it+1} \\ \beta_{it} = (c_{i1t}, c_{i2t}, c_{i3t}, c_{i4t}, y_{1t}) \\ v_{t+1} \sim N(0, \mathcal{I}_5 * 0.05^2) \\ \epsilon_{it+1} \sim t_5(0, 0.05^2) \end{cases}$$

where c_{ijt} represents the j-th feature of firm i at time t, while y_{jt} represents the j-th macroeconomic variable at time t. Excess returns of firm i at time t are modeled as a function of x_{it}, which includes all 94 firm characteristics and the 8 macro variables, plus an error term, which is in turn defined as a 5x1 vector of factors along with an epsilon term to account for idiosyncratic errors. This setup ensures that the simulated returns will always differ by a random amount from the true returns implied by the DGP.

I define four specifications of the f^* function to observe how methods differ in their predictive ability depending on the underlying model.

Case 1 is a simple linear model. The predictor set x_{it} contains 94 firm characteristics c_{it} plus 8 macro variables y_t. WLOG, I assume returns are exclusively a function of the first 4 firm features and the first macro variable [14]

$$f^*(x_{it}) = c_{i1t} + c_{i2t} + c_{i3t} + c_{i4t} + y_{1t}$$

[14]Since I am working with simulated data, the features are only numbered and have no economic interpretation.

Case 2 is a simple non-linear model, which allows for the interaction between firm characteristics. All variables enter non-linearly and there is also a discrete variable, i.e., the sign of the interaction between two firm characteristics.

$$f^*(x_{it}) = c_{i1t} \cdot c_{i2t} + c_{i3t} \cdot c_{i4t} + c_{i4t}^2 + y_{1t}^2 + sgn(c_{i1t} \cdot c_{i5t})$$

Case 3 and 4 are both non-linear models that allow for interactions between firm characteristics and macro variables. These are the most complex specifications which most likely provide a better description of reality. The difference between the two models lies in the set of regressors: while model 3 contains the same set as the two previous specifications (94 regressors + 8 macro variables), model 4 has a much richer set of covariates given by the firm characteristics plus their interaction with the macro variables, for a total of $94 + 94 \times 8 = 846$ regressors. Model 4 follows closely what is done in Gu, Kelly and Xiu (2020). Due to technological and computational constraints, I will not be able to include such a large number of covariates when working with real data. Thus, using simulated data, I aim to determine whether the inclusion of interaction terms in the set of regressors is justified by a significantly higher R^2. On the other hand, if model 3 and 4 exhibit highly correlated outputs, both qualitatively and in magnitude, working with only the "raw" covariates is preferred as it drastically reduces computational times, giving also a clearer economic interpretation.

$$f^*(x_{it}) = c_{i1t} \cdot y_{2t} + c_{i3t} \cdot y_{4t} + c_{i4t} \cdot c_{i3t}^2 + y_{1t}^2 \cdot c_{i2t} + sgn(y_{1t} \cdot c_{i5t})$$

To enhance clarity, Table 1 summarizes the four models.

Table 1: Summary of simulation models

Model	$f^*(x_{it})$	N of Regressors
1	$f^*(x_{it}) = c_{i1t} + c_{i2t} + c_{i3t} + c_{i4t} + y_{1t}$	102
2	$f^*(x_{it}) = c_{i1t} \cdot c_{i2t} + c_{i3t} \cdot c_{i4t} + c_{i4t}^2 + y_{1t}^2 + sgn(c_{i1t} \cdot c_{i5t})$	102
3	$f^*(x_{it}) = c_{i1t} \cdot y_{2t} + c_{i3t} \cdot y_{4t} + c_{i4t} \cdot c_{i3t}^2 + y_{1t}^2 \cdot c_{i2t} + sgn(y_{1t} \cdot c_{i5t})$	102
4	$f^*(x_{it}) = c_{i1t} \cdot y_{2t} + c_{i3t} \cdot y_{4t} + c_{i4t} \cdot c_{i3t}^2 + y_{1t}^2 \cdot c_{i2t} + sgn(y_{1t} \cdot c_{i5t})$	846

***Note**: This table reports the 4 models used with simulated data, along with the number of regressors. Model 1 is linear, model 2 is non-linear, model 3 allows for the interaction between firm characteristics and macroeconomic variables, which model 4 also adds to the set of regressors.*

Other assumptions have to be made to simulate the dataset. As frequently done in the literature, such as in Gu, Kelly and Xiu (2020) and Freyberger, Neuhierl and Weber (2020), firm characteristics are simulated using the cross section rank function:

$$c_{ijt} = \frac{2}{N+1} CSrank(\overline{c}_{ijt}) - 1, \qquad \overline{c}_{ijt} = \rho_j \overline{c}_{ijt-1} + \epsilon_{ijt}$$
$$\rho_j \sim U[0.9, 1], \qquad \epsilon_{ijt} \sim N(0, 1 - \rho_j^2)$$

This ensures that characteristics exhibit a strong degree of persistence and are cross-sectionally ranked in each period and mapped in the [-1,1] interval. This is useful since many of the methods I will employ require normalization of the data. In addition, the absolute value of a characteristic is not of particular relevance in this analysis, as what matters more is the relative size of characteristics in the cross-section. This is especially important when dealing with a panel dataset than spans over 60 years. The magnitude of certain variables loses meaning over time: just think of the size, cash holdings or corporate investment variables.

The macro variables are instead modeled as a simple AR(1) with a high degree of persistence.

$$y_{jt} = 0.95\, y_{jt-1} + u_{jt}$$
$$u_{jt} \sim N(0, 0.1)$$

I consider 200 firms and a period of 210 months, thus $N = 200$ and $T = 210$. This setup aims to strike a good balance between computational efficiency and the representativeness of the sample. The dimensions of the dataset are therefore 42000×102 in the first three cases, and 42000×846 in the fourth. I split the data in three equal parts for training, validation and testing. Appendix E presents all parameter combinations tested in the validation set for each method. To achieve feasible computational times for each simulation, I do not employ a very granular tuning of the hyperparameters, never testing more than 7-8 combinations for each method. The time saved here will allow for more granular tuning when working with real data.

I simulate 30 samples using the fixed estimation scheme. In each simulation, I use the training sample to estimate the four models using the methods described in the previous section. I then use the validation sample for hyperparameter tuning, i.e., to find the set of hyperparameters that has the strongest out-of-sample performance in the validation

sample. Finally, I use the testing sample to evaluate the predictive performance of each model and method. Performance is assessed using both the out-of-sample coefficient of determination R^2_{oos} and the mean squared error (MSE). R^2_{oos} is defined as follows:

$$R^2_{oos} = 1 - \frac{\sum_{(i,t)\in Testing}(r_{i,t+1} - \hat{r}_{i,t+1})^2}{\sum_{(i,t)\in Testing} r^2_{i,t+1}}$$

It is a tweaked version of the traditional coefficient of determination. The numerator represents the residual sum of squares, computed in the testing sample only. The denominator, however, consists of the sum of squared actual returns in the testing sample. This occurs as the out-of-sample R^2 is comparing the predictions of an elaborate model that uses covariate information with those of a benchmark (null) model, which in this case assumes a naive forecast of zero for all predicted returns. Using this benchmark is common in stock return forecasting. Another popular benchmark choice is the historical average, which forecasts future returns as the average returns from the training sample. This was first used in Campbell and Thompson (2008).

Table 2 presents the average R^2, both in and out-of-sample, of the 30 simulations.

Table 2: Results of simulations

Estimation method	Model 1		Model 2		Model 3		Model 4	
R^2(%)	IS	OoS	IS	OoS	IS	OoS	IS	OoS
OLS	14.79	0.24	9.75	-0.36	8.83	-3.73	32.15	-17.39
PLS	12.15	1.32	8.04	1.43	7.54	-2.15	17.94	-3.46
PCR	4.88	0.53	3.19	1.27	2.08	-0.27	5.21	-0.06
Elastic Net	10.22	3.07	4.95	3.65	4.05	0.10	11.58	0.03
Lasso	9.16	3.07	2.28	3.96	2.02	0.28	6.01	0.54
LGBM	11.45	2.54	5.40	4.37	2.96	0.45	5.69	0.71
XGBoost	8.27	2.35	5.67	4.50	4.53	0.21	6.00	0.76
NN	3.92	-0.57	1.41	1.47	1.27	-1.09	1.89	-2.50

***Note**: This table reports the in-sample (IS) and out-of-sample (OoS) R^2 for each model and method. Results are in percentages.*

For what concerns model 1, Lasso and Elastic Net deliver the highest R^2_{oos}, closely followed by LGBM and XGBoost. These two non-linear methods consistently rank among the best performers across all models. Their relative simplicity seems to come through in these small Montecarlo samples. On the contrary, and as expected, the linear methods work best in the first (and second) model, while their performance deteriorates in the last two. The negative values of R^2_{oos} indicate that the method is underperforming compared to a naive forecast of zero.

Another compelling finding is the mediocre performance of neural networks, which display negative R^2_{oos} in models 1, 3 and 4. I argue this is mainly stemming from the small training sample. Neural networks are heavily parametrized and need large samples to become efficient and avoid overfitting. I am thus expecting the performance of NNs to reach the level of the other non-linear methods, if not surpass them, when dealing with real data.

OLS is clearly the worst model in all settings except the first one. PLS beats PCR in the first two models, but is then outperformed in the more advanced settings. Variable selection methods consistently achieve a higher R^2_{oos} than dimensionality-reduction methods.

Focusing on the comparison between Model 3 and 4, the results are qualitatively similar. Model 4 has consistently a larger in-sample R^2, indicating that the inclusion of all the interaction terms in the set of regressors leads to higher explained variability of returns. However, regarding out-of-sample performance, model 4 has a greater variability of results: the worse-performing methods (e.g., OLS and NN) have a poorer performance while LGBM and XGBoost perform somewhat better. As the analysis focuses on the best-performing methods, it can be concluded that, when working with real data without including the interaction terms, the obtained results can be considered a conservative estimate of the predictive performance of non-linear methods.

Table 3 presents the average MSE of the simulations.

Table 3: Results of simulations continued

Estimation method	Model 1		Model 2		Model 3		Model 4	
MSE	IS	OoS	IS	OoS	IS	OoS	IS	OoS
OLS	0.00917	0.01079	0.00937	0.01103	0.00941	0.01109	0.00697	0.01404
PLS	0.00946	0.01049	0.00956	0.01065	0.00955	0.01074	0.00845	0.01105
PCR	0.01028	0.01065	0.01090	0.01099	0.01051	0.01041	0.01037	0.01034
Elastic Net	0.00967	0.01014	0.00987	0.01025	0.00991	0.01028	0.00909	0.01038
Lasso	0.00980	0.01013	0.01015	0.01020	0.01013	0.01021	0.00969	0.01020
LGBM	0.00956	0.01028	0.00982	0.01015	0.01003	0.01019	0.00972	0.01017
XGBoost	0.00989	0.01029	0.00980	0.01014	0.00986	0.01023	0.00970	0.01016
NN5	0.01035	0.01090	0.01024	0.01053	0.01018	0.01043	0.01011	0.01049

***Note**: This table reports the in-sample (IS) and out-of-sample (OoS) mean squared error (MSE) for each model and method.*

This table is more difficult to read. However, it gives the same results as the above one. Indeed, there is almost perfect negative comovement between R^2_{oos} and MSE, with a correlation coefficient of -0.92.

5 Empirical analysis

I now turn to the analysis of real data. I begin by presenting the predictive performance of the machine learning methods using the fixed estimation scheme. These are reported in table 4.

Table 4: Predictive performance with fixed estimation scheme

$R^2(\%)$	OLS	FF5	PLS	PCR	ENet	Lasso	LGBM	XGB	NN5
IS	1.80	0.04	1.74	0.08	0.26	0.27	3.18	7.69	1.25
OoS	-1.63	0.20	-0.09	-0.06	0.34	0.34	0.14	0.15	0.21

***Note**: This table reports the in-sample (IS) and out-of-sample (OoS) R^2 for each machine learning method, using a fixed estimation window. Results are in percentages.*

As expected, OLS is the worst performing model, showing a negative out-of-sample R^2. While the "traditional" R^2 can only take values in the $[0, 1]$ interval, R^2_{oos} has a codomain of $(-\infty, 1]$. A negative R^2_{oos} implies that the elaborate model, e.g., OLS, underperforms the zero forecast, while a positive value suggests superior predictive performance. In this case, the interpretation of R^2_{oos} is the proportion of the mean squared error loss of the benchmark model that is explained by the elaborate model, as stated in Hawinkel et al. (2024).

The dimensionality-reduction models, PLS and PCR, also struggle to perform well out of sample. On the contrary, penalized versions of OLS deliver significantly better results. A possible explanation lies in the structure of the regressors. As shown in the correlation heatmap, there does not seem to be strong multicollinearity in the dataset, as only few pairs of variables exhibit strong correlation. This may suggest that the main flaw of the dataset is the presence of easily-detectable redundant variables, which can be effectively filtered out by using variable selection methods. On the other hand, dimensionality-reducing techniques may struggle to efficiently summarize the dataset comprised of relatively independent covariates.

All non-linear methods perform well. LGBM and XGBoost display similar performance. The feed-forward neural network achieves a slightly higher performance. I report only the results of the best version, which has 5 hidden layers, in contrast with the findings in Gu, Kelly and Xiu (2020). Regarding the other tested neural networks, a general trend that emerges is the slightly better performance of wider but shallower networks compared to deeper but narrower ones, which somehow goes against the argument in Eldan and Shamir (2016).

When comparing the real data results with the simulation models, there is a rather close correspondence between the real results and models 3 and 4 (with the exception of neural networks), which allowed for interactions between firm characteristics and macro variables. It appears that a data-generating process with interactions, even if they are not explicitly included in the set of regressors (as in model 3), is able to predict the relative performance of different methods quite accurately in real data. This reflects the quality of the assumptions made in the construction of the DGP.

Finally, the low magnitude of the predictive results should not be discouraging. It

has to be remembered that these figures reflect the monthly predictive performance of the model. As Fama and French (1988) demonstrate, the R^2 statistic is increasing in the forecasting horizon when the predictor variables are persistent. In addition, Campbell and Thompson (2008) prove that even a small R^2_{oos} is economically meaningful, and that an excessively large value should be deemed to good to be true.[15]

Table 5 presents the out-of-sample results for the recursive estimation scheme.

Table 5: Predictive performance with recursive estimation scheme

R^2(%)	OLS	FF5	PLS	PCR	ENet	Lasso	LGBM	XGB	NN5	Ensemble
OoS	-1.25	0.23	-0.23	0.05	0.33	0.36	0.48	0.45	0.49	0.35

***Note**: This table reports the out-of-sample (OoS) R^2 for each method, as well as for an ensemble model that takes the average of all forecasts, using a recursive estimation window. Results are in percentages and improve those of table 4.*

With the exception of PLS, most methods show improved results in the recursive setting, particularly the non-linear ones. XGBoost has its performance increase by more than three times. This suggests that allowing for complex predictor interactions is rewarded in the recursive setting. Non-linear models benefit the most from the yearly refit of the model and the expanding training set, as their flexibility allows them to better capture new information, and the non-linearities and shifting patterns it may contain. In the recursive setting, LGBM and NN5 have a slight edge on the other methods, imposing themselves as the best-performing methods.

While non-linear methods continuously learn from the recursive approach, the performance of variable selection methods remains unaltered. This result can also be easily rationalized as in the new setting the relationship between the predictors and the target variable may still remain sufficiently stable for Lasso and Elastic Net to keep selecting the same variables, thus not benefitting from new data. The main difference between EN

[15]According to this paper, the magnitude of R^2_{oos} should be compared with the squared Sharpe ratio of all analyzed stocks. A relatively large R^2 suggests the possibility to exploit the information in the predictive returns of the model to increase the average monthly return of the portfolio by a factor that is approximately equal to the ratio between R^2_{oos} and the squared Sharpe ratio.

and Lasso lies in the number of selected variables. While Lasso always selects between 4 and 6 predictors in the final models, Elastic Net selects between 18 and 25. This is explained by the difference in the penalty function, as Elastic Net combines Lasso (L1) and Ridge (L2) penalties, resulting in less aggressive feature selection, as some coefficients are shrunk instead of being set to zero.

The variables that are consistently selected by both Lasso and Elastic Net are the 1-month momentum (mom1) and cash productivity (cashpr). XGBoost and Lgbm agree on the same factors. In addition, they prioritize investment-related variables, such as corporate investment (cinvest) and capital expenditures and inventory (invest), as well as performance and risk measures, such as return on assets (roaq) and return volatility (retvol).

Given the overall improvement in predictive performance, I will rely on the recursive estimation scheme for the remainder of the analysis, which will focus on assessing the economic relevance of these findings.

5.1 Portfolio creation

A compelling application of the results is to investigate whether the predictive performance of the machine learning models can be translated into a portfolio that outperforms the market index. The most common approach (Gu, Kelly and Xiu 2020, Drobertz and Otto, 2021) is to construct zero-cost, decile-based portfolios sorted by expected returns. Each month, all stocks in the testing sample are ranked according to their predicted returns for the following month. The stocks are then grouped into deciles, and a portfolio that buys the top decile and shorts the bottom decile is constructed. In the following periods, stocks are again ranked and the portfolio is adjusted to always include the highest and lowest predicted stocks.

The performance of the portfolio is assessed by tracking the actual returns of the selected stocks, and is evaluated using different metrics. The Sharpe ratio is perhaps the most common performance metric used. It measures the risk-adjusted return of a portfolio by dividing excess returns by the standard deviation of the portfolio's excess

returns.

$$SR = \frac{\mathbb{E}_t(R^p - R^f)}{\sigma_t(R^p)}$$

It is interpreted as the excess returns generated per unit of risk.

Another important decision involves the selection of portfolio weights, as portfolio returns are computed as a weighted sum of the individual stock returns:

$$R^p_{t+1} = \sum_{i=1}^{n} w^p_{i,t} \cdot r_{i,t+1}$$

The choice of weights can significantly impact portfolio performance. In general, the literature commonly considers two weighting schemes: equal weights (EW) and value weights (VW). The former assigns the same weight to all stocks in the portfolio. The latter determines weights based on the relative market capitalization of the stock. Thus, in a value-weighted portfolio, stocks of large companies will receive much higher weights than smaller stocks. The advantage of using value weights is that they better reflect the overall market, neglecting small stocks that often have higher trading frictions. Value-weighted portfolios are also more scalable, as large funds cannot invest substantial amounts in small, illiquid stocks without impacting their price.

Nevertheless, I will not use value weights for several reasons. Firstly, applying value weights leads to a highly skewed weight distribution. On average, five stocks account for more than 50% of the weights, in a portfolio that contains hundreds of stocks. This excessive concentration of risk in few large stocks creates an idiosyncratic risk that I consider too large and that goes outside the scope of my analysis. Indeed, I am interested in exploiting the power of machine learning methods precisely so that I can construct portfolios which are exposed to a broad set of stocks that are fairly represented. In addition, the choice of a more uniform distribution of weights better resonates with the objective function used in model estimation. Models are estimated by minimizing equal-weighted forecast errors, so it makes sense to consider a distribution of weights that is not heavily skewed. As I am not discriminating stocks in the statistical part of the analysis, it is coherent not to excessively discriminate them in the economic application as well.

In addition to equal weights, I construct volatility weighted portfolios, where stock weights are inversely proportional to their volatility in the previous period. The dataset contains the regressor "return volatility" which was shown in Ang et al. (2006) to be neg-

atively associated with excess returns. While still providing a rather uniform distribution of weights, such weights attempt to mitigate the high volatility risk in equal-weighted portfolios. These may rely too heavily on small stocks, which tend to be more volatile. A volatility-weighted portfolio should reduce risk exposure while preserving broad diversification.

Table 6 shows the time series average of predicted and actual portfolio excess returns for each model, along with their standard deviation and the annualized[16] Sharpe Ratio.

Table 6: Performance of Equal-weighted and Volatility-weighted Portfolios

	Equal-weighted				Volatility-weighted			
	Pred [%]	Real [%]	Sd [%]	SR	Pred [%]	Real [%]	Sd [%]	SR
OLS								
Low (L)	−0.59	0.34	6.52	0.18	−0.48	0.57	5.05	0.39
2	0.80	0.72	5.59	0.44	0.80	0.70	4.13	0.59
3	1.33	0.78	5.25	0.51	1.30	0.66	3.88	0.59
4	1.71	0.84	5.14	0.57	1.61	0.61	3.75	0.56
5	2.02	0.83	5.01	0.57	1.93	0.62	3.66	0.59
6	2.30	0.81	4.92	0.57	2.23	0.72	3.66	0.68
7	2.61	0.83	5.09	0.56	2.53	0.73	3.83	0.66
8	2.99	0.91	5.50	0.57	2.96	0.81	4.22	0.67
9	3.56	1.12	5.93	0.65	3.52	0.93	4.75	0.68
High (H)	5.11	1.59	7.29	0.75	4.98	1.31	5.88	0.77
H-L	5.70	1.25	3.53	1.22	5.46	0.73	3.17	0.80
FF5								
Low (L)	0.93	0.65	6.25	0.36	0.93	0.78	5.17	0.52
2	1.06	0.92	5.56	0.57	1.06	0.94	4.83	0.67
3	1.08	0.87	5.33	0.56	1.08	0.79	4.23	0.65
4	1.10	0.75	5.35	0.49	1.08	0.65	3.92	0.57
5	1.10	0.73	4.90	0.52	1.03	0.51	3.17	0.55
6	1.10	0.59	4.20	0.48	1.02	0.42	2.66	0.55
7	1.11	0.78	5.33	0.51	1.10	0.70	3.91	0.62
8	1.12	1.06	5.90	0.62	1.12	0.97	5.07	0.66
9	1.14	1.24	6.41	0.67	1.14	1.14	5.60	0.71
High (H)	1.27	1.16	7.13	0.56	1.27	1.14	6.25	0.63
H-L	0.34	0.51	2.80	0.63	0.34	0.36	2.51	0.50

[16]The annualized Sharpe Ratio is computed by multiplying the monthly SR by $\sqrt{12}$, i.e., the square root of the number of months in a year.

	Equal-weighted				Volatility-weighted			
	Pred [%]	Real [%]	Sd [%]	SR	Pred [%]	Real [%]	Sd [%]	SR
PLS								
Low (L)	−1.10	0.33	6.39	0.18	−1.00	0.56	5.22	0.37
2	0.14	0.75	5.55	0.47	0.15	0.73	4.32	0.58
3	0.62	0.79	5.15	0.53	0.61	0.63	3.72	0.59
4	0.95	0.81	5.03	0.56	0.94	0.64	3.67	0.61
5	1.22	0.83	4.92	0.58	1.16	0.64	3.64	0.61
6	1.49	0.88	4.90	0.62	1.42	0.74	3.70	0.69
7	1.79	0.79	5.19	0.53	1.74	0.76	4.00	0.65
8	2.17	0.88	5.50	0.56	2.15	0.80	4.30	0.65
9	2.74	1.17	6.29	0.65	2.71	0.98	5.05	0.67
High (H)	4.23	1.53	7.43	0.72	4.10	1.25	5.97	0.73
H-L	5.33	1.20	3.76	1.11	5.09	0.69	3.37	0.71
PCR								
Low (L)	0.66	0.86	8.74	0.34	0.67	1.02	7.78	0.45
2	0.88	0.95	6.68	0.49	0.87	1.07	6.12	0.60
3	0.97	0.85	5.98	0.49	0.96	0.89	5.38	0.57
4	1.04	0.84	5.45	0.54	1.03	0.84	4.87	0.60
5	1.10	0.90	5.02	0.62	1.02	0.79	4.14	0.66
6	1.15	0.79	4.50	0.61	1.09	0.65	3.45	0.65
7	1.21	0.65	4.13	0.54	1.20	0.52	3.00	0.60
8	1.30	0.83	4.33	0.66	1.29	0.74	3.07	0.83
9	1.49	0.91	5.23	0.60	1.46	0.69	3.59	0.66
High (H)	2.04	1.17	6.63	0.61	2.00	1.02	5.42	0.65
H-L	1.39	0.31	4.64	0.23	1.33	0.01	4.44	0.00
ENet								
Low (L)	−0.60	0.40	6.23	0.22	−0.52	0.66	5.28	0.43
2	0.15	0.72	5.34	0.47	0.15	0.81	4.37	0.64
3	0.42	0.82	5.00	0.57	0.42	0.72	3.78	0.66
4	0.60	0.86	4.85	0.62	0.60	0.65	3.70	0.61
5	0.75	0.87	4.87	0.62	0.72	0.76	3.84	0.69
6	0.90	0.87	4.98	0.60	0.87	0.75	3.94	0.66
7	1.06	0.88	5.16	0.59	1.02	0.71	4.14	0.59
8	1.27	0.96	5.80	0.58	1.24	0.84	4.80	0.61
9	1.59	0.93	6.40	0.50	1.57	0.87	5.44	0.55
High (H)	2.35	1.45	8.16	0.62	2.28	1.09	6.87	0.55
H-L	2.95	1.06	4.71	0.78	2.80	0.44	4.62	0.33

	Equal-weighted				Volatility-weighted			
	Pred [%]	Real [%]	Sd [%]	SR	Pred [%]	Real [%]	Sd [%]	SR
Lasso								
Low (L)	−0.15	0.44	6.26	0.24	−0.08	0.63	5.50	0.40
2	0.49	0.75	5.20	0.50	0.49	0.85	4.32	0.68
3	0.69	0.82	4.84	0.59	0.69	0.80	3.91	0.71
4	0.82	0.83	4.87	0.59	0.81	0.73	3.89	0.65
5	0.92	0.89	4.80	0.65	0.90	0.82	3.88	0.73
6	1.02	0.87	4.91	0.61	0.97	0.71	3.99	0.62
7	1.14	0.92	5.22	0.61	1.10	0.81	4.31	0.65
8	1.28	0.91	5.80	0.55	1.25	0.76	4.92	0.53
9	1.49	0.88	6.42	0.47	1.46	0.79	5.63	0.49
High (H)	2.00	1.43	8.53	0.58	1.96	1.12	7.43	0.52
H-L	2.15	0.99	4.88	0.70	2.03	0.49	4.85	0.35
Lgbm								
Low (L)	−0.38	0.40	6.97	0.20	−0.36	0.68	6.07	0.39
2	0.19	0.66	5.98	0.38	0.19	0.73	5.17	0.49
3	0.35	0.68	5.29	0.45	0.33	0.63	4.29	0.51
4	0.44	0.87	4.81	0.63	0.42	0.77	3.78	0.71
5	0.51	0.64	4.74	0.47	0.51	0.42	3.74	0.39
6	0.58	0.89	4.82	0.64	0.57	0.84	3.78	0.77
7	0.67	0.99	5.00	0.69	0.67	0.84	3.90	0.75
8	0.83	1.00	5.39	0.65	0.80	0.88	4.29	0.71
9	1.19	0.99	5.92	0.58	1.16	0.78	4.59	0.59
High (H)	2.47	1.62	7.96	0.70	2.11	1.09	5.87	0.64
H-L	2.84	1.22	3.58	1.18	2.46	0.41	2.78	0.51
XGB								
Low (L)	−1.14	0.42	6.63	0.22	−1.08	0.71	5.64	0.43
2	−0.44	0.65	5.49	0.41	−0.43	0.78	4.41	0.61
3	−0.18	0.66	5.17	0.44	−0.17	0.54	4.12	0.45
4	−0.01	0.82	4.69	0.61	−0.01	0.60	3.67	0.56
5	0.12	0.70	4.81	0.50	0.09	0.60	3.82	0.55
6	0.27	0.83	5.19	0.55	0.25	0.70	4.31	0.57
7	0.46	0.93	5.53	0.58	0.47	0.80	4.44	0.63
8	0.75	0.99	5.52	0.62	0.74	0.92	4.63	0.69
9	1.17	1.07	6.04	0.61	1.17	1.00	4.76	0.73
High (H)	2.85	1.69	7.76	0.75	2.46	1.14	5.49	0.72
H-L	3.99	1.26	3.68	1.19	3.54	0.44	2.84	0.53

	Equal-weighted				Volatility-weighted			
	Pred [%]	Real [%]	Sd [%]	SR	Pred [%]	Real [%]	Sd [%]	SR
NN								
Low (L)	−2.47	0.32	7.20	0.16	−2.36	0.46	6.22	0.26
2	−0.53	0.68	6.26	0.38	−0.57	0.71	5.19	0.48
3	−0.17	0.83	5.52	0.52	−0.16	0.71	4.47	0.55
4	0.13	0.80	5.04	0.55	0.12	0.63	3.89	0.56
5	0.41	0.96	4.86	0.69	0.42	0.69	3.82	0.63
6	0.52	0.89	4.69	0.66	0.53	0.78	3.46	0.78
7	0.69	0.84	4.95	0.59	0.68	0.72	3.70	0.68
8	0.81	0.94	4.96	0.65	0.81	0.91	3.61	0.87
9	0.97	0.95	5.72	0.58	0.98	0.92	4.50	0.71
High (H)	1.31	1.53	7.35	0.72	1.21	1.32	6.10	0.75
H-L	3.78	1.20	3.33	1.25	3.57	0.85	3.52	0.84

***Note**: This table reports the performance of the equal-weighted and volatility-weighted portfolios for each method and decile. H-L stands for high-minus-low, which is the portfolio strategy used. For each portfolio, the table reports predicted returns, actual returns, the standard deviation of actual returns, and the annualized Sharpe ratio for each decile.*

The deciles are well sorted. While the middle deciles are sometimes not monotonically increasing (though this is not critical for the analysis), the key result is the strong separation between the highest and lowest deciles.

Another finding that is consistent across all methods is the higher performance of the equal-weighted portfolio. Realized excess returns and Sharpe Ratios are more pronounced. This may indicate that results in the equal-weighted portfolio are in large part driven by more volatile stocks. Indeed, volatility-weighted portfolios display lower volatility at the cost of lower realized returns, which combined lead to lower Sharpe ratios. Furthermore, the realized returns of the bottom decile, i.e., the shorted component of the portfolio, are never negative. This implies that simply buying the highest decile will yield higher returns, but at the cost of higher risk. Indeed, the returns of the top decile are also the most volatile, which is expected since large returns are typically associated with higher risk. Therefore, the High-Low portfolio generally offers the best risk-adjusted performance as measured by the Sharpe Ratio.

Non-linear models are the best-performing ones. They have similar Sharpe Ratios in the equal-weighted portfolios, while in the volatility-weighted the neural network has an

edge over gradient boosting methods. Principal components regression is significantly the worst method in terms of economic gains. This leads to a compelling discussion concerning the mapping between predictive and economic performance.

Leitch and Tanner (1991) already showed a weak association between performance-related statistics and economic profitability. These results somehow seem to go in the same direction. For instance, OLS yields a H-L portfolio that has a higher Sharpe Ratio than the other linear models, despite producing the lowest R^2_{oos}. When constructing decile-based portfolios, what matters the most is the model's ability in properly ranking stock returns. It is important to forecast which stocks will yield very large returns and which will yield very low returns. But the precision of the forecast is not of utmost relevance, what matters is to select the correct stock. Being able to better explain the cross-section of returns is obviously desired, but does not necessarily translate into creating a more profitable portfolio.

Another mapping that is not necessarily monotonic is the one between Sharpe Ratios and cumulative returns, which will be shortly assessed.

Figures 5 and 6 plot the cumulative log returns for all models, for both equal and volatility-weighted portfolios, compared against those of the S&P 500.

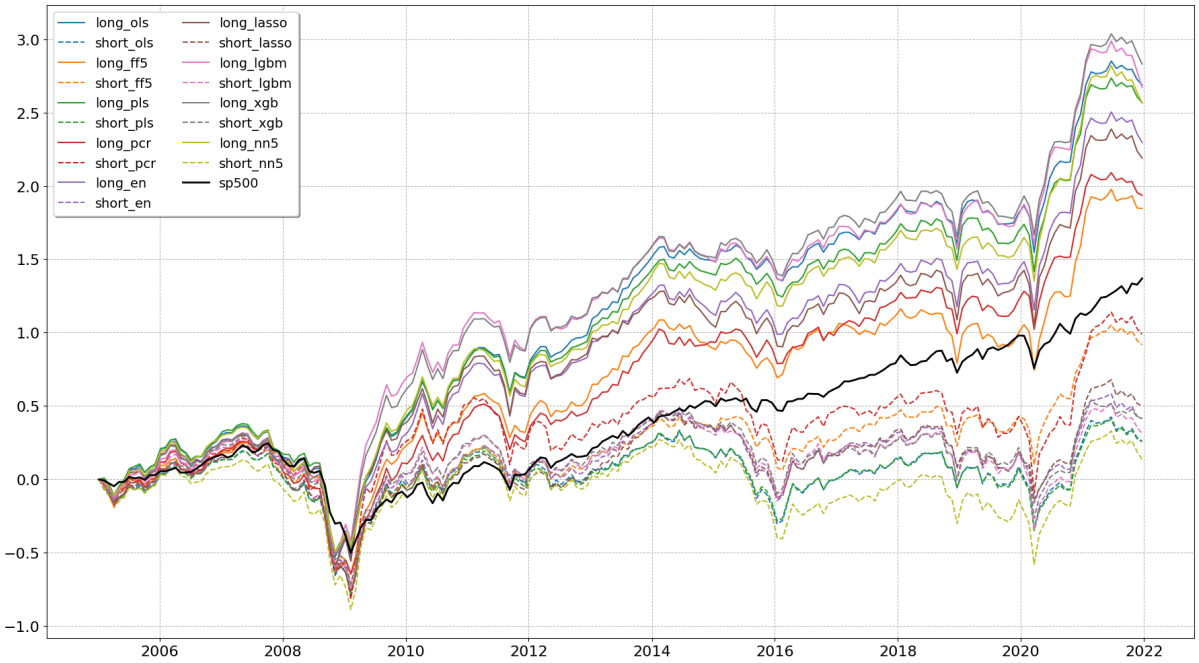

Figure 5: Cumulative log returns of equal-weighted portfolios

***Note**: This table reports the cumulative log returns of the equal-weighted portfolio for each method. The solid lines represent the long component of the portfolio, the dashed lines the short component.*

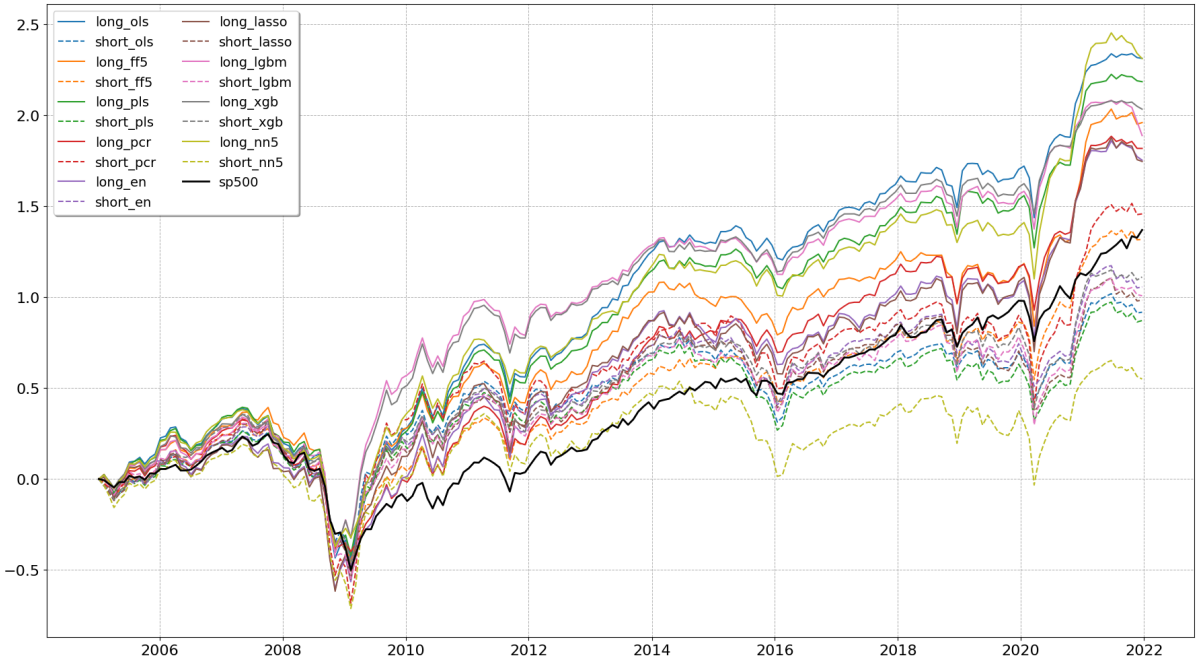

Figure 6: Cumulative log returns of volatility-weighted portfolios

***Note**: This table reports the cumulative log returns of the volatility-weighted portfolio for each method. The solid lines represent the long component of the portfolio, the dashed lines the short component. Methods have the same colors as in Figure 5.*

XGBoost and LGBM deliver the highest cumulative log returns among equal-weighted portfolios, with their long components reaching a value of nearly 3, which is more than double the value reached by the market index (1.25). The outperformance of the machine learning models is even more striking when considering cumulative returns in levels. The returns of XGBoost and LGBM long portfolio are almost six times higher than those of the S&P 500. In general, all models outperform the market index in the equal-weighted scenario. The Fama-French 5-factor model and PCR are the worst performers. The short portofolios correctly do not make high returns, which allow for a profitable H-L strategy.

On the other hand, in the volatility-weighted setting the 5-layer neural network stands out as the best-performing model, having also the worst performing short component. While all other models significantly outperform the market index with their long components, the main issue arises from abnormally high returns of the short components. For instance, FF5 and PCR have short components that yield higher returns than the market index itself. In general, the performance of the volatility-weighted portfolios is lower in magnitude than equal-weighted. The neural network still has cumulative returns that

double those of the S&P 500 in logs.

The results align with Gu, Kelly and Xiu (2020) in showing the superiority of non-linear models, with Lewellen (2015) in showing that parsimonious versions of OLS perform well, and with Drobertz and Otto (2021) in finding that plain OLS is better than some more sophisticated models and that PCR is the worst performing model, yielding much lower results than the comparable PLS.

The mapping between Sharpe Ratios and cumulative returns is more or less preserved. This should not be taken for granted as one does not imply the other. A portfolio can exhibit high cumulative returns combined with high volatility, which yields a lower Sharpe Ratio than a more stable portfolio with lower returns. The Sharpe Ratio is much more informative than cumulative returns as a metric, as it contains valuable information concerning the variance of the portfolio, which is not reflected in cumulative returns.

To get a sense of the performance of the models, Table 7 presents the annualized returns of the high-minus-low portfolio for XGBoost with equal weights and NN5 with volatility weights, the best-performing models in their respective categories.

Table 7: Annualized returns of S&P 500, XGB, and NN5 portfolios.

Year	XGB_EW	NN5_VW	S&P 500	Year	XGB_EW	NN5_VW	S&P 500
2005	0.0643	0.1217	0.0300	2014	-0.0132	0.0549	0.1139
2006	-0.0462	0.0598	0.1362	2015	0.3729	0.2504	-0.0073
2007	-0.0057	-0.0314	0.0353	2016	0.0569	-0.0208	0.0954
2008	0.0366	0.0210	-0.3849	2017	0.1013	0.0635	0.1942
2009	0.6356	0.0263	0.2345	2018	0.0014	0.0803	-0.0624
2010	0.2578	0.0981	0.1278	2019	-0.0127	-0.1146	0.2888
2011	0.1772	0.0983	0.0000	2020	0.8269	0.7616	0.1626
2012	0.1106	0.1062	0.1341	2021	0.1964	0.2532	0.2689
2013	0.1804	0.0672	0.2960	Total	10.32	3.98	3.43

***Note**: This table reports the annualized returns of the H-L, decile-based portfolios for the best model for each type of weight. These are XGBoost for equal weights and 5-layer neural network for volatility weights. They are compared with returns of the S&P 500. Returns have to be multiplied by 100 to become percentages.*

What is most striking is that both models rarely experience negative yearly returns. The largest loss for the equal-weighted portfolio occurred in 2006, with a modest decline of -4.6%. Remarkably, the portfolio still generated positive returns during major crises, such as an impressive 83% return in 2020, a year dominated by the COVID-19 pandemic, when the S&P 500 gained only 16%. The same holds for the volatility-weighted portfolio, albeit with lower returns in magnitude. This portfolio yields returns of 76% in 2020. Both portfolios perform extremely well during the Great Financial Crisis when compared to the market index, which lost nearly 40% in 2008.

However, the results have to be taken with some caution. In the end, the analysis is based on historical data that has already occured. Assessing the performance of an investment strategy on past data is commonly known as backtesting. While out-of-sample forecasting reduces the risk of backtesting overfitting, the strategy remains vulnerable to non-stationary market conditions and survivorship bias. To address survivorship bias, I included firms that were once listed but are no longer present on the stock exchanges. In addition, I choose not to dwell excessively on the dates and length of the training, validation and testing sample, so as to be less subject to the risk of backtest overfitting.

Following Bailey et al. (2014), I disclose that only one backtest was performed. Initially, I attempted a robustness check with different sample splits, but this proved too computationally expensive. For instance, I tried increasing the testing sample by one year while reducing the validation sample, which resulted in slightly higher out-of-sample performance.

Since cumulative returns of all portfolios seem to follow the same path with difference in levels, it may be interesting to compute the correlation among portfolios. Some methods may select the same stocks in large proportions in the top and bottom decile. To measure the degree of similarity in the long and short components of the portfolios, I employ the Jaccard Index. It is defined as the ratio of the intersection of two sets to their union:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

In this context, sets A and B represent the stocks in the top or bottom decile for models A and B, respectively. A high Jaccard Index indicates high similarity in portfolio composition between two models. By computing it for each pair of models, both for the top

and bottom decile, one can retrieve a similarity matrix, that serves as a proxy for the correlation between portfolios.

Table 8 reports the Jaccard Index for each model. The Index is computed with monthly frequency and the average is reported.

Table 8: Top and bottom decile portfolio correlation (Jaccard Index)

Bottom/Top	OLS	FF5	PLS	PCR	EN	Lasso	LGBM	XGB	NN5
OLS	—	0.13	0.73	0.16	0.39	0.27	0.24	0.24	0.31
FF5	0.10	—	0.13	0.15	0.12	0.09	0.11	0.11	0.12
PLS	0.72	0.11	—	0.16	0.46	0.33	0.25	0.26	0.30
PCR	0.16	0.09	0.16	—	0.14	0.10	0.12	0.13	0.14
EN	0.39	0.08	0.45	0.13	—	0.64	0.29	0.28	0.25
Lasso	0.30	0.06	0.35	0.10	0.67	—	0.27	0.25	0.20
LGBM	0.18	0.07	0.19	0.10	0.20	0.20	—	0.52	0.20
XGB	0.20	0.07	0.21	0.12	0.22	0.22	0.43	—	0.20
NN5	0.26	0.08	0.25	0.14	0.22	0.21	0.16	0.17	—

***Note**: The upper triangle of the matrix shows the Jaccard Index for the top decile portfolios, while the lower triangle shows the Jaccard Index for the bottom decile. The diagonal has been replaced by a black line for separation.*

The results are strikingly similar among the top and bottom decile. This may hint at symmetry in predicting returns, with models that assign high expected returns to similar stocks also having high similarity in the worst performing stocks. PCR and FF5, the poorest performing models, are also those whose stocks are least similar to other models. OLS and PLS display similar stock selection, as well as the two variable selection methods and the gradient boosting methods.

Another interesting statistic regards portfolio autocorrelation, which measures how much each portfolio is readjusted on a monthly basis. This provides insights into the stability of each portfolio in terms of stock selection. A high degree of autocorrelation implies that the portfolio is relatively stable, with little turnover from month to month.

Table 9 shows the average "autocorrelation" for each portfolio, proxied again by using the Jaccard index. This measure can be considered a close representation of the inverse

of portfolio turnover.

Table 9: Average portfolio autocorrelation

	OLS	FF5	PLS	PCR	ENet	Lasso	LGBM	XGB	NN
Avg Autocorrelation	0.31	0.88	0.28	0.83	0.19	0.15	0.31	0.38	0.39

***Note**: This table reports the average portfolio autocorrelation for each method. It is computed using the Jaccard Index. The interpretation is the following: on average, 69% of the stocks in the OLS portfolio are changed each month.*

Interestingly, the two worst-performing models exhibit the highest level of autocorrelation. FF5 and PCR are too stable to effectively capture the dynamic interactions of the data. They are defeating the purpose of machine learning models, which are supposed to efficiently extract signals from new data and adapt the portfolio composition accordingly. FF5 is based on a few well-known factors that do not have high time variation, which explains its tendency to select the same stocks each month.

On the other hand, variable selection methods are characterized by the lowest autocorrelation (highest turnover). Their aggressive feature selection implies that they will extensively change the portfolio composition. Non-linear models exhibit an intermediate level of autocorrelation.

5.2 Centile-based portfolios

On average, a month in the testing sample (January 2005 - December 2021) contains around 6000 stocks. Thus, the decile-based portfolios constructed earlier require holding 600 stocks in the long component and shorting another 600 each month. This is a cumbersome and impractical strategy for the average investor and even for most funds. The large dimensions of the dataset could be further exploited to construct more granular portfolios.

Therefore, in this section I construct centile-based portfolios, which are constructed using the top and bottom centiles of predicted returns. This approach is not widely used in the literature, but I believe it can represent a valid option when dealing with a dataset of such large dimensions. With centile-based portfolios, the long component

would contain "only" 60 stocks each month, as well as the short component. Creating this portfolio also goes in the direction of making this analysis more implementable by the average investor. It is interesting to observe whether these portfolios can deliver higher risk-adjusted returns compared to decile-based portfolios.

For this purpose, I consider the best-performing model in each weight category. Table 10 shows the performance of the centile-based portfolio using equal weights based on XGBoost predictions, and the volatility-weighted centile-based portfolio, based on the 5-layer neural network. As there are too many centiles to be displayed, the results show only the bottom and top centiles, as well as the high-minus-low (H-L) portfolio.

Table 10: Performance of centile-based Portfolios

	XGB Equal-weighted				NN5 Volatility-weighted			
	Pred [%]	Real [%]	Sd [%]	SR	Pred [%]	Real [%]	Sd [%]	SR
Low (L)	−2.33	−0.13	8.76	−0.05	−4.02	−0.37	7.78	−0.17
High (H)	7.80	4.72	14.50	1.13	6.30	2.58	10.35	0.86
H-L	10.13	4.86	11.28	1.49	10.32	2.95	7.30	1.40

***Note**: The left-hand side of the table reports the performance of the equal-weighted, centile-based portfolio using XGBoost. The right-hand side reports the performance of the volatility-weighted, centile-based portfolio using a 5-layer neural network. H-L stands for high-minus-low, which is the portfolio strategy used. For each portfolio, the table reports predicted returns, actual returns, the standard deviation of actual returns and the annualized Sharpe ratio.*

Centile-based portfolios display a significantly higher performance than decile-based ones. In this setting, the Sharpe ratio of the H-L portfolio with equal weights increases from 1.19 to 1.49, while that of the volatility-weighted portfolio jumps from 0.84 to 1.40. Moreover, the bottom centile displays negative realized returns as desired, making the long component strictly dominated by the H-L portfolio, which simultaneously has higher retuns and lower volatility. This did not occur in the decile-based portfolios.

It appears that focusing on the extreme centiles allows to remove some noisy, misclassified predictions that entered the top and bottom deciles. As expected, predicted and actual returns are more pronounced. This comes with a higher standard deviation of returns. However, despite this increased volatility, the higher Sharpe ratios indicate that realized returns more than compensate for the additional risk.

Figure 7 plots cumulative log returns of the centile-based portfolios, compared against those of the S&P 500.

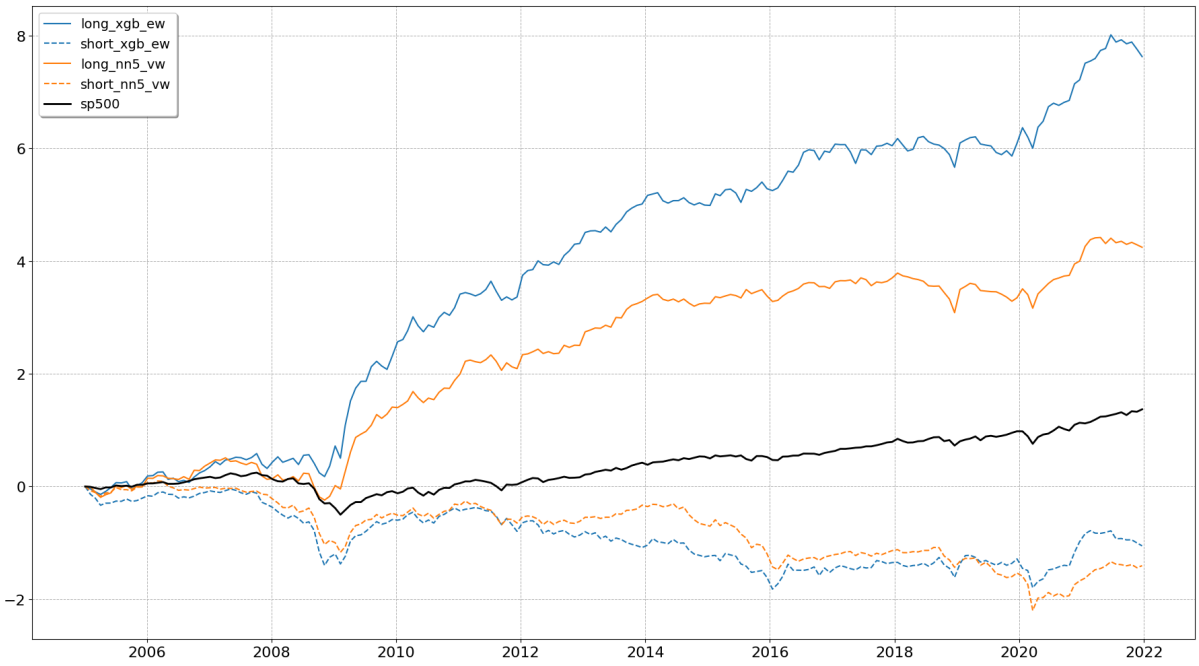

Figure 7: Cumulative log returns of centile-based portfolios

***Note**: This table reports the cumulative log returns of the equal-weighted portfolio for XGBoost and the volatility-weighted portfolio for the neural network. Both are centile-based. The solid lines represent the long component of the portfolio, the dashed lines the short component. The black line shows the cumulative log returns of the S&P 500.*

The graph helps to visualize the improved performance of centile-based portfolios. XGBoost increases its cumulative log returns of the long component from 3 to nearly 8, while the neural network improves from 2.5 to over 4.

In summary, the higher concentration of high-quality signals in the extreme centiles allows the portfolios to deliver better risk-adjusted performance, justifying the move towards more granular portfolio construction.

6 Conclusion

In this paper, I conduct a comparative analysis of several machine learning methods, employing both fixed and recursive estimation schemes. The latter yields higher predictive performance. Non-linear methods achieve the highest R^2_{oos}, followed by variable selection methods. Dimensionality-reduction methods are the poorest performers, together

with plain OLS. Predictive performance is translated into economic profitability, which is assessed by constructing zero-cost decile-based portfolios. Non-linear methods are still the best-performing ones. In particular, XGBoost generates the most profitable portfolio with equal weights, and a 5-layer neural network yields the best portfolio with volatility weights. These portfolios, when compared to the market index, perform especially well in periods of crisis. Interestingly, some methods that are outperformed by others in terms of predictive performance generate instead higher Sharpe ratios, suggesting the absence of a clear, one-to-one mapping between predictive and economic performance. Non-linear methods have intermediate turnover values, while traditional methods either adjust portfolios too aggressively or remain too stable. Finally, switching from decile to centile-based portfolios significantly improves risk-adjusted performance.

The findings of this study confirm the potential of machine learning methods to create profitable trading strategies. Investors and portfolio managers could greatly benefit from using these methods for stock selection.

The results should also be treated as a conservative estimate of the potential of non-linear models. My analysis has been limited by computational constraints. Being able to work with an even richer dataset and with a more granular set of hyperparameters can only improve the magnitude and quality of the results. Nonetheless, the power of this study lies also in showing that even an ill-equipped investor can implement machine learning methods effectively.

The future of this research area is promising. New non-linear methods will continue to emerge. Advances in technology will further reduce computational costs. Exploring more advanced neural networks, such as generative adversarial networks, could capture even more interactions between the multitude of factors that are starting to be disentangled. To overcome the problems around the mapping between predictive and economic performance, an interesting approach could be that of conducting hyperparameter tuning based on economic performance. For instance, the selected set of hyperparameters would be the one that yields the portfolio with the highest Sharpe ratio in the validation sample.

References

Amengual, D. and Watson, M. W. (2007). Consistent estimation of the number of dynamic factors in a large n and t panel. *Journal of Business & Economic Statistics*, 25(1):91–96.

Ang, A., Hodrick, R. J., Xing, Y., and Zhang, X. (2006). The cross-section of volatility and expected returns. *The journal of finance*, 61(1):259–299.

Bagnara, M. (2024). Asset pricing and machine learning: a critical review. *Journal of Economic Surveys*, 38(1):27–56.

Bai, J. and Ng, S. (2002). Determining the number of factors in approximate factor models. *Econometrica*, 70(1):191–221.

Bailey, D. H., Borwein, J. M., de Prado, M. L., and Zhu, Q. J. (2014). Pseudomathematics and financial charlatanism: The effects of backtest over fitting on out-of-sample performance. *Notices of the AMS*, 61(5):458–471.

Banz, R. W. (1981). The relationship between return and market value of common stocks. *Journal of financial economics*, 9(1):3–18.

Basu, S. (1977). Investment performance of common stocks in relation to their price-earnings ratios: A test of the efficient market hypothesis. *The journal of Finance*, 32(3):663–682.

Campbell, J. Y. (1987). Stock returns and the term structure. *Journal of financial economics*, 18(2):373–399.

Campbell, J. Y. and Shiller, R. J. (1988). The dividend-price ratio and expectations of future dividends and discount factors. *The review of financial studies*, 1(3):195–228.

Campbell, J. Y. and Thompson, S. B. (2008). Predicting excess stock returns out of sample: Can anything beat the historical average? *The Review of Financial Studies*, 21(4):1509–1531.

Carhart, M. M. (1997). On persistence in mutual fund performance. *The Journal of finance*, 52(1):57–82.

Chen, R., Pelger, M., and Zhu, J. (2019). Deep learning in asset pricing. *Available at SSRN 3350138*. Forthcoming in *Management Science*.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Cowles 3rd, A. (1933). Can stock market forecasters forecast? *Econometrica: Journal of the Econometric Society*, pages 309–324.

Drobetz, W. and Otto, T. (2021). Empirical asset pricing via machine learning: evidence from the european stock market. *Journal of Asset Management*, 22(7):507–538.

Eldan, R. and Shamir, O. (2016). The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR.

Fama, E. F. (1970). Efficient capital markets. *Journal of finance*, 25(2):383–417.

Fama, E. F. and Blume, M. E. (1966). Filter rules and stock-market trading. *The Journal of Business*, 39(1):226–241.

Fama, E. F. and French, K. R. (1988). Dividend yields and expected stock returns. *Journal of financial economics*, 22(1):3–25.

Fama, E. F. and French, K. R. (1992). The cross-section of expected stock returns. *Journal of Finance*, 47(2):427–465.

Fama, E. F. and French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, 33(1):3–56.

Fama, E. F. and French, K. R. (2015). A five-factor asset pricing model. *Journal of financial economics*, 116(1):1–22.

Fama, E. F. and French, K. R. (2020). The value premium. *Review of Asset Pricing Studies*, 10(1):1–25.

Fama, E. F. and MacBeth, J. D. (1973). Risk, return, and equilibrium: Empirical tests. *Journal of political economy*, 81(3):607–636.

Feng, G., Giglio, S., and Xiu, D. (2020). Taming the factor zoo: A test of new factors. *The Journal of Finance*, 75(3):1327–1370.

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

Giglio, S., Kelly, B., and Xiu, D. (2022). Factor models, machine learning, and asset pricing. *Annual Review of Financial Economics*, 14(1):337–368.

Green, J., Hand, J. R., and Zhang, X. F. (2017). The characteristics that provide independent information about average us monthly stock returns. *The Review of Financial Studies*, 30(12):4389–4436.

Gu, S., Kelly, B., and Xiu, D. (2020). Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273.

Guo, H. (2006). On the out-of-sample predictability of stock market returns. *The Journal of Business*, 79(2):645–670.

Hawinkel, S., Waegeman, W., and Maere, S. (2024). Out-of-sample r 2: Estimation and inference. *The American Statistician*, 78(1):15–25.

Ioffe, S. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jensen, M. C. and Benington, G. A. (1970). Random walks and technical theories: Some additional evidence. *The Journal of finance*, 25(2):469–482.

Kelly, B. and Pruitt, S. (2013). Market expectations in the cross-section of present values. *The Journal of Finance*, 68(5):1721–1756.

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

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

Leitch, G. and Tanner, J. E. (1991). Economic forecast evaluation: profits versus the conventional error measures. *The American Economic Review*, pages 580–590.

Lettau, M. and Ludvigson, S. (2001). Consumption, aggregate wealth, and expected stock returns. *the Journal of Finance*, 56(3):815–849.

Lewellen, J. (2015). The cross section of expected stock returns. *Critical Finance Review*, 4(1):1–44.

Lintner, J. (1965). The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. *Review of Economics and Statistics*, 47(1):13–37.

Łoś, H., Mendes, G. S., Cordeiro, D., Grosso, N., Costa, H., Benevides, P., and Caetano, M. (2021). Evaluation of xgboost and lgbm performance in tree species classification with sentinel-2 data. In *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pages 5803–5806. IEEE.

Masters, T. (1993). *Practical neural network recipes in C++*. Morgan Kaufmann.

Moghar, A. and Hamiche, M. (2020). Stock market prediction using lstm recurrent neural network. *Procedia computer science*, 170:1168–1173.

Mossin, J. (1966). Equilibrium in a capital asset market. *Econometrica: Journal of the econometric society*, pages 768–783.

Qiu, M., Song, Y., and Akagi, F. (2016). Application of artificial neural network for the prediction of stock market returns: The case of the japanese stock market. *Chaos, Solitons & Fractals*, 85:1–7.

Rapach, D. and Zhou, G. (2013). Forecasting stock returns. In *Handbook of economic forecasting*, volume 2, pages 328–383. Elsevier.

Rossi, A. G. (2018). Predicting stock market returns with machine learning. *Georgetown University*.

Sharpe, W. F. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. *The journal of finance*, 19(3):425–442.

Welch, I. and Goyal, A. (2008). A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies*, 21(4):1455–1508.

Yang, Y., Wu, Y., Wang, P., and Jiali, X. (2021). Stock price prediction based on xgboost and lightgbm. In *E3s web of conferences*, volume 275, page 01040. EDP Sciences.

Ye, J., Goswami, B., Gu, J., Uddin, A., and Wang, G. (2024). From factor models to deep learning: Machine learning in reshaping empirical asset pricing. *arXiv preprint arXiv:2403.06779*.

Zhang, Y. (2022). Stock price prediction method based on xgboost algorithm. In *2022 International Conference on Bigdata Blockchain and Economy Management (ICBBEM 2022)*, pages 595–603. Atlantis Press.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320.

Appendix

A. List of regressors

Table 11: Details of the Characteristics

No.	Acronym	Firm characteristic	Author(s)	Frequency
1	absacc	Absolute accruals	Bandyopadhyay, Huang & Wirjanto (2010)	Annual
2	acc	Working capital accruals	Sloan (1996)	Annual
3	aeavol	Abnormal earnings announcement volume	Lerman, Livnat & Mendenhall (2007)	Quarterly
4	age	# years since first Compustat coverage	Jiang, Lee & Zhang (2005)	Annual
5	agr	Asset growth	Cooper, Gulen & Schill (2008)	Annual
6	baspread	Bid-ask spread	Amihud & Mendelson (1989)	Monthly
7	beta	Beta	Fama & MacBeth (1973)	Monthly
8	betasq	Beta squared	Fama & MacBeth (1973)	Monthly
9	bm	Book-to-market	Rosenberg, Reid & Lanstein (1985)	Annual
10	bm_ia	Industry-adjusted book to market	Asness, Porter & Stevens (2000)	Annual
11	cash	Cash holdings	Palazzo (2012)	Quarterly
12	cashdebt	Cash flow to debt	Ou & Penman (1989)	Annual
13	cashpr	Cash productivity	Chandrashekar & Rao (2009)	Annual
14	cfp	Cash flow to price ratio	Desai, Rajgopal & Venkatachalam (2004)	Annual
15	cfp_ia	Industry-adjusted cash flow to price ratio	Asness, Porter & Stevens (2000)	Annual
16	chatoia	Industry-adjusted change in asset turnover	Soliman (2008)	Annual
17	chcsho	Change in shares outstanding	Pontiff & Woodgate (2008)	Annual
18	chempia	Industry-adjusted change in employees	Asness, Porter & Stevens (1994)	Annual
19	chinv	Change in inventory	Thomas & Zhang (2002)	Annual
20	chmom	Change in 6-month momentum	Gettleman & Marks (2006)	Monthly
21	chpmia	Industry-adjusted change in profit margin	Soliman (2008)	Annual
22	chtx	Change in tax expense	Thomas & Zhang (2011)	Quarterly
23	cinvest	Corporate investment	Titman, Wei & Xie (2004)	Quarterly
24	convind	Convertible debt indicator	Valta (2016)	Annual
25	currat	Current ratio	Ou & Penman (1989)	Annual
26	depr	Depreciation / PP&E	Holthausen & Larcker (1992)	Annual
27	divi	Dividend initiation	Michaely, Thaler & Womack (1995)	Annual
28	divo	Dividend omission	Michaely, Thaler & Womack (1995)	Annual
29	dolvol	Dollar trading volume	Chordia, Subrahmanyam & Anshuman (2001)	Monthly
30	dy	Dividend to price	Litzenberger & Ramaswamy (1982)	Annual
31	ear	Earnings announcement return	Kishore, Brandt, Santa-Clara & Venkatachalam (2008)	Quarterly
32	egr	Growth in common shareholder equity	Richardson, Sloan, Soliman & Tuna (2005)	Annual
33	ep	Earnings to price	Basu (1977)	Annual
34	gma	Gross profitability	Novy-Marx (2013)	Annual
35	grCAPX	Growth in capital expenditures	Anderson & Garcia-Feijoo (2006)	Annual

No.	Acronym	Firm characteristic	Author(s)	Frequency
36	grltnoa	Growth in long term net operating assets	Fairfield, Whisenant & Yohn (2003)	Annual
37	herf	Industry sales concentration	Hou & Robinson (2006)	Annual
38	hire	Employee growth rate	Bazdresch, Belo & Lin (2014)	Annual
39	idiovol	Idiosyncratic return volatility	Ali, Hwang & Trombley (2003)	Monthly
40	ill	Illiquidity	Amihud (2002)	Monthly
41	indmom	Industry momentum	Moskowitz & Grinblatt (1999)	Monthly
42	invest	Capital expenditures and inventory	Chen & Zhang (2010)	Annual
43	lev	Leverage	Bhandari (1988)	Annual
44	lgr	Growth in long-term debt	Richardson, Sloan, Soliman & Tuna (2005)	Annual
45	maxret	Maximum daily return	Bali, Cakici & Whitelaw (2011)	Monthly
46	mom12m	12-month momentum	Jegadeesh (1990)	Monthly
47	mom1m	1-month momentum	Jegadeesh & Titman (1993)	Monthly
48	mom36m	36-month momentum	Jegadeesh & Titman (1993)	Monthly
49	mom6m	6-month momentum	Jegadeesh & Titman (1993)	Monthly
50	ms	Financial statement score	Mohanram (2005)	Quarterly
51	mvel1	Size (market value)	Banz (1981)	Monthly
52	mve_ia	Industry-adjusted size	Asness, Porter & Stevens (2000)	Annual
53	nincr	Number of earnings increases	Barth, Elliott & Finn (1999)	Quarterly
54	operprof	Operating profitability	Fama & French (2015)	Annual
55	orgcap	Organizational capital	Eisfeldt & Papanikolaou (2013)	Annual
56	pchcapx_ia	Industry adjusted % change in capital expenditures	Abarbanell & Bushee (1998)	Annual
57	pchcurrat	% change in current ratio	Ou & Penman (1989)	Annual
58	pchdepr	% change in depreciation	Holthausen & Larcker (1992)	Annual
59	pchgm_pchsale	% change in gross margin - % change in sales	Abarbanell & Bushee (1998)	Annual
60	pchquick	% change in quick ratio	Ou & Penman (1989)	Annual
61	pchsale_pchinvt	% change in sales - % change in inventory	Abarbanell & Bushee (1998)	Annual
62	pchsale_pchrect	% change in sales - % change in A/R	Abarbanell & Bushee (1998)	Annual
63	pchsale_pchxsga	% change in sales - % change in SG&A	Abarbanell & Bushee (1998)	Annual
64	pchsaleinv	% change sales-to-inventory	Ou & Penman (1989)	Annual
65	pctacc	Percent accruals	Hafzalla, Lundholm & Van Winkle (2011)	Annual
66	pricedelay	Price delay	Hou & Moskowitz (2005)	Monthly
67	ps	Financial statements score	Piotroski (2000)	Annual
68	quick	Quick ratio	Ou & Penman (1989)	Annual
69	rd	R&D increase	Eberhart, Maxwell & Siddique (2004)	Annual
70	rd_mve	R&D to market capitalization	Guo, Lev & Shi (2006)	Annual
71	rd_sale	R&D to sales	Guo, Lev & Shi (2006)	Annual
72	realestate	Real estate holdings	Tuzel (2010)	Annual
73	retvol	Return volatility	Ang, Hodrick, Xing & Zhang (2006)	Monthly
74	roaq	Return on assets	Balakrishnan, Bartov & Faurel (2010)	Quarterly
75	roavol	Earnings volatility	Francis, LaFond, Olsson & Schipper (2004)	Quarterly
76	roeq	Return on equity	Hou, Xue & Zhang (2015)	Quarterly

No.	Acronym	Firm characteristic	Author(s)	Frequency
77	roic	Return on invested capital	Brown & Rowe (2007)	Annual
78	rsup	Revenue surprise	Kama (2009)	Quarterly
79	salecash	Sales to cash	Ou & Penman (1989)	Annual
80	saleinv	Sales to inventory	Ou & Penman (1989)	Annual
81	salerec	Sales to receivables	Ou & Penman (1989)	Annual
82	secured	Secured debt	Valta (2016)	Annual
83	securedind	Secured debt indicator	Valta (2016)	Annual
84	sgr	Sales growth	Lakonishok, Shleifer & Vishny (1994)	Annual
85	sin	Sin stocks	Hong & Kacperczyk (2009)	Annual
86	sp	Sales to price	Barbee, Mukherji, & Raines (1996)	Annual
87	std_dolvol	Volatility of liquidity (dollar trading volume)	Chordia, Subrahmanyam & Anshuman (2001)	Monthly
88	std_turn	Volatility of liquidity (share turnover)	Chordia, Subrahmanyam & Anshuman (2001)	Monthly
89	stdacc	Accrual volatility	Bandyopadhyay, Huang & Wirjanto (2010)	Quarterly
90	stdcf	Cash flow volatility	Huang (2009)	Quarterly
91	tang	Debt capacity/firm tangibility	Almeida & Campello (2007)	Annual
92	tb	Tax income to book income	Lev & Nissim (2004)	Annual
93	turn	Share turnover	Datar, Naik & Radcliffe (1998)	Monthly
94	zerotrade	Zero trading days	Liu (2006)	Monthly

***Note**: This table lists all 94 firm characteristics, along with the acronym used in the dataset, the paper that found them to be significant in explaining stock returns, and their frequency.*

Table 12: Macroeconomic variables

No.	Acronym	Macro variable	Definition	Frequency
1	dp	Dividend-price ratio	Difference between the log of dividends and the log of prices	Monthly
2	ep	Earnings-price ratio	Difference between the log of earnings and the log of prices	Monthly
3	bm	Book-to-market ratio	Ratio of book value to market value	Monthly
4	ntis	Net equity expansion	Ratio of 12-month moving sums of net issues by NYSE listed stocks divided by the total end-of-year market capitalization of NYSE stocks	Monthly
5	tbl	Treasury-bill rate	3- Month Treasury Bill: Secondary Market Rate	Monthly
6	tms	Term spread	Difference between the long term yield on government bonds and the T-bill	Monthly
7	dfy	Default spread	Difference between BAA and AAA-rated corporate bond yields	Monthly
8	svar	Stock variance	Sum of squared daily returns on the S&P 500	Monthly

***Note**: This table lists the 8 macroeconomic variables, along with the acronym used in the dataset, their definition (as stated in Welch and Goyal, 2008), and their frequency.*

B. Correlation heatmap of firm characteristics

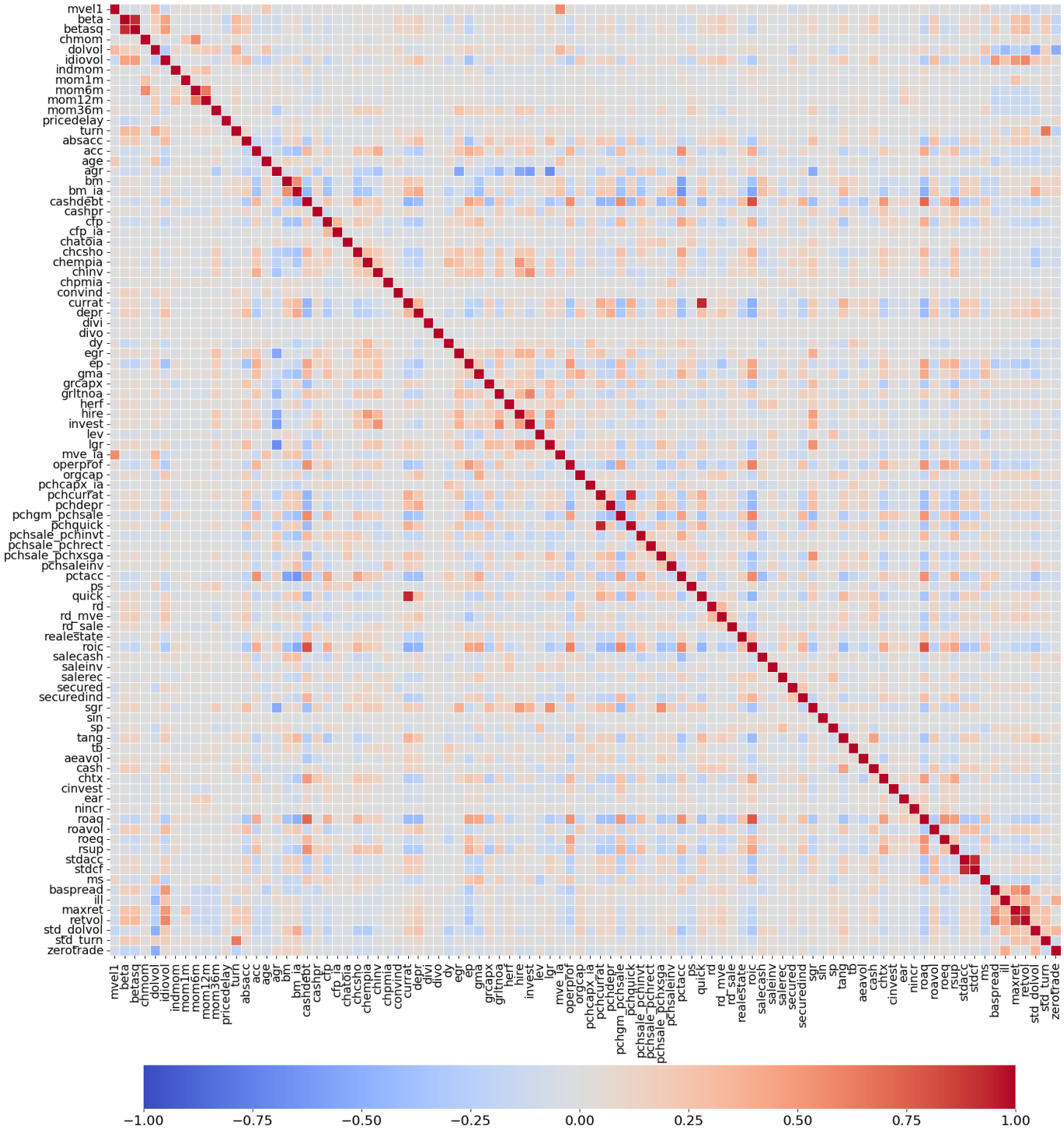

Figure 8: Correlation heatmap of the 94 firm characteristics

***Note**: This table reports the correlation for each pair of regressors. Characteristics are sorted in alphabetical order. Red indicates positive correlation, blue indicates negative correlation. Greyish color indicates correlation around 0.*

C. Gradient boosting algorithm

Consider a generic differentiable loss function $L(y_i, F(x))$, where $F(x)$ is a function of the covariates that gives the predicted returns.

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\text{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

γ is the initial prediction that minimizes the loss function.
If the loss function is $L(y_i, \gamma) = \frac{1}{2}(y_i - \gamma)^2$, then $\gamma = \bar{y}$.

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, \ldots, n$
M denotes the number of trees. Compute the pseudo-residual for each observation (actual minus prediction from previous tree).

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$
The terminal regions are the leaves, indexed by j.

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\text{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$
For each leaf, compute the output value that minimizes the loss function with updated with the previous prediction and only considers observations in that leaf.

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
Make a new prediction equal to the previous one plus the scaled output value.

Step 3: Output $F_M(x)$.

D. Regularization of neural networks

- Batch normalization (Ioffe, 2015). Applied to layers, it normalizes the output of the activation function for each batch (subset of dataset). Used to prevent internal covariate shift, i.e., the distribution of the inputs changing at each layer, hence speeding up the convergence process resulting in faster computations.

- Early stopping. Stops the optimization algorithm when the prediction error in the validation sample starts to increase, regardless of whether it has been minimized in the training sample.

- Learning rate shrinkage. The learning rate is shrunk as the gradient gets closer to 0, to prevent the model from making adjustments that are too large that may jump past the minimum, leading to slower convergence.

- Ensembles. Consider models with different random seeds and combine their outputs to achieve more robust and generalizable findings.

E. Hyperparameters

Table 14: Tested hyperparameters in simulations

OLS	-		
PLS	$N \in \{1, 3, 5, 10, 50\}$		
PCR	$N \in \{1, 3, 5, 10, 50\}$		
EN	$\lambda \in \{0.0001, 0.001, 0.002, 0.004\}$		
Lasso	$\lambda \in \{0.0001, 0.001, 0.002, 0.004\}$		
LGBM	Depth $= 1 \sim 2$	$K \in \{10, 50, 100, 200, 500\}$	$LR \in \{0.01, 0.1\}$
XGBoost	Depth $= 1$	$K \in \{500, 600\}$	$LR = 0.01$
NN	Layers $= 1 \sim 3$	$l1 \in \{1e-5, 1e-3\}$	$LR \in \{0.01, 0.1\}$
	Batch size $= 280$	Epochs $= 100$	Patience $= 5$

***Note**: N stands for the number of factors in the model, λ controls regularization strength, Depth is the number of layers in the regression tree, K is the number of trees, LR is the learning rate, Layers is the number of hidden layers, $l1$ is a regularization term in the optimization of the parameters, Batch size is the number of datapoints processed at each iteration, Epochs is the number of times the neural network goes through the dataset, and Patience is the number of epochs after which, if the model is not improving, the training will stop.*

Table 15: Tested hyperparameters in the empirical section

OLS	-			
FF5	-			
PLS	$N \in \{1, 2, 3, 4, 5, 10, 25, 50\}$			
PCR	$N \in \{1, 2, 3, 4, 5, 10, 25, 50\}$			
EN	$\lambda \in \{0.0001, 0.001, 0.002, 0.004, 0.01\}$			
Lasso	$\lambda \in \{0.0001, 0.001, 0.002, 0.004, 0.01\}$			
LGBM	Depth $= 1 \sim 3$	$K \in \{10, 50, 100, 200, 500, 1000\}$	$LR \in \{0.01, 0.1\}$	
XGBoost	Depth $= 1 \sim 3$	$K \in \{200, 500, 600, 800\}$	$LR \in \{0.01, 0.1\}$	
NN	Layers $= 1 \sim 5$	$l1 \in \{1e-5, 1e-4, 1e-3\}$	$LR \in \{0.01, 0.1\}$	
	Batch size $= 20000$	Epochs $= 200$	Patience $= 5$	

***Note**: N stands for the number of factors in the model, λ controls regularization strength, Depth is the number of layers in the regression tree, K is the number of trees, LR is the learning rate, Layers is the number of hidden layers, $l1$ is a regularization term in the optimization of the parameters, Batch size is the number of datapoints processed at each iteration, Epochs is the number of times the neural network goes through the dataset, and Patience is the number of epochs after which, if the model is not improving, the training will stop.*