Empirical Asset Pricing via Machine Learning*

Shihao Gu† Bryan Kelly‡ Dacheng Xiu§

University of Chicago Booth School of Business¶

This Version: December 7, 2017

Abstract

We synthesize the field of machine learning with the canonical problem of empirical asset pricing: Measuring asset risk premia. We use the widely understood empirical setting of predicting the time series and cross section of stock (and portfolio) returns to perform a comparative analysis of methods in the machine learning repertoire, including generalized additive models, boosted regression trees, random forests, and neural networks. At the broadest level, we find that machine learning has great promise for describing asset price behavior. Our implementation establishes a new standard for accuracy in measuring risk premia summarized by unprecedented high out-of-sample return prediction R^2. We identify the best performing methods (trees and neural nets) and trace their predictive gains to allowance of non-linear predictor interactions that are missed by other methods. Lastly, we find that *all* methods agree on the same small set of dominant predictive signals that are variations on momentum, liquidity, and market beta. Improved risk premia measurement through machine learning can simplify the investigation into economic mechanisms of asset pricing and justifies its growing role in innovative financial technologies.

Key words: Fintech, Machine Learning, Return Prediction, Cross-Section of Returns, Ridge Regression, (Group) Lasso, Elastic Net, Random Forest, Gradient Boosting, Deep Neural Networks

*We benefited from discussions with Rob Engle, Lasse Pedersen, Guofu Zhou, and seminar and conference participants at Fannie Mae, Tsinghua Workshop on Big Data and Internet Economics, the 2017 Conference on Financial Predictability and Data Science.

†Email: shihao.gu@chicagobooth.edu.

‡Email: bryan.kelly@chicagobooth.edu.

§Email: dacheng.xiu@chicagobooth.edu.

¶Address: 5807 S Woodlawn Avenue, Chicago, IL 60637, USA.

1 Introduction

Machine learning methods are emerging as ubiquitous technologies throughout the financial ecosystem. Fintech startups are applying machine learning to problems as diverse as portfolio management (for example, Binatix), microfinance and loan approval (LendingClub and ZestFinance) and fraud detection (BillGuard), and large financial institutions are investing heavily in machine learning and other fintech architectures.[1] Despite the rapidly growing deployment of machine learning in the fintech industry, exploration of these methods in academic finance research is barely nascent.

In this article, we conduct a comparative analysis of a machine learning methods for finance. We do so in the context of perhaps the most widely studied problem in finance, that of measuring equity risk premia.

1.1 Primary Contributions

Our primary contributions are two-fold. First, we provide a new benchmark of accuracy in measuring risk premia at the aggregate market and individual stock levels. The accuracy of our risk premium estimates is summarized by unprecedented high out-of-sample predictive R^2's that are robust across a variety of machine learning specifications.

Return prediction is economically meaningful. The fundamental goal of asset pricing is to understand the behavior risk premia. If expected returns were perfectly observed, we would still need theories to describe their behavior and empirical analysis to test those theories. But risk premia are notoriously difficult to measure—the very nature of efficient markets leads return variation to be driven by unforecastable news. Our research highlights gains that can be achieved in prediction and identifies the most informative predictor variables. This helps resolve the problem of risk premium measurement, which then facilitates more reliable investigation into economic mechanisms of asset pricing.

Second, we synthesize the empirical asset pricing literature with the field of machine learning. Machine learning methods allow us to be more expansive in the set of potential predictor variables and in specifications of functional form that we consider, relative to traditional asset pricing methods. It is this flexibility that helps us push the frontier of risk premium measurement. Our synthesis is also a pedagogical contribution. Interest in machine learning methods for finance has grown tremendously in both academia and industry.This article provides a comparative overview of machine learning methods applied to the two canonical problems of empirical asset pricing: predicting returns in the cross section and time series. Our view is that the best way for a finance researcher to become acquainted with machine learning is to apply and compare the performance of each of its methods in a familiar empirical problem.

[1]The industry magazine *Banking Technology* reports that the top ten US banks by assets under management have invested in 56 fintech companies between 2012 and 2017Q1, of which seven are data analytics specialists.

1.2 What is Machine Learning?

The definition of "machine learning" is inchoate and is often context specific. We use the term to describe (i) a diverse collection of high-dimensional models for statistical prediction, combined with (ii) so-called "regularization" methods for model selection and mitigation of overfit, and (iii) efficient algorithms for searching among model specifications.

The high-dimensional nature of machine learning methods (element (i) of this definition) enhances their flexibility relative to more traditional econometric prediction techniques. This flexibility brings hope of better approximating the unknown and likely complex data generating process underlying equity risk premia. With enhanced flexibility, however, comes a higher propensity of overfitting the data. Element (ii) of the machine learning definition describes refinements in implementation that emphasize stable out-of-sample performance to explicitly guard against overfit. Finally, with many predictors it becomes infeasible to exhaustively traverse and compare all model permutations. Element (iii) describes clever tools in the machine learning arsenal for arriving at the optimal specification with minimal computational cost.

1.3 Why Apply Machine Learning to Asset Pricing?

A number of aspects of empirical asset pricing make this field particularly attractive for analysis with machine learning methods.

1) Two main research agendas have monopolized modern empirical asset pricing research. The first seeks to describe and understand differences in expected returns across assets. The second focuses on dynamics of the aggregate market equity risk premium. Because it is a conditional expectation of a future realized excess return, measurement of an asset's risk premium is fundamentally a problem of prediction. Machine learning, whose methods are largely specialized for prediction tasks, is thus ideally suited to the problem of risk premium measurement.

2) The collection of conditioning variables for the risk premium is large. The profession has accumulated a staggering list of alternative predictors argued by researchers to possess forecasting power for returns. The number of stock-level predictive characteristics reported in the literature numbers in the hundreds and macroeconomic predictors of the aggregate market number in the dozens.[2] Additionally, predictors are often close cousins and highly correlated. Traditional prediction methods break down when the predictor count approaches the observation count or predictors are highly correlated. Because it emphasizes variable selection and dimension reduction techniques to reduce degrees of freedom and strip out redundant variation among predictors, machine learning is well suited for precisely these kinds of challenging prediction problems.

3) Further complicating the problem is ambiguity regarding functional forms through which the high-dimensional predictor set might affect risk premia. Should they enter linearly? If non-linearities

[2] Green et al. (2013) count 330 stock-level predictive signals in published or circulated drafts. Harvey et al. (2016) study 316 "factors," which include firm characteristics and common factors, for describing stock return behavior. They note that this is only a subset of those studied in the literature. Welch and Goyal (2008) analyze nearly 20 predictors for the aggregate market return. In both stock and aggregate return predictions, there presumably exists a much larger set of predictors that were tested but failed to predict returns and were thus never reported.

are needed, which form should they take? Must we consider interactions among predictors? Such questions rapidly proliferate the set of potential model specifications. The theoretical literature offers little guidance for winnowing the list conditioning variables and functional forms. Four aspects of machine learning make it well suited for problems of ambiguous functional form. The first is its diversity. As a suite of dissimilar methods it casts a wide net in its specification search. Second, some methods, such as generalized linear models and neural networks, are explicitly designed to approximate complex non-linear associations. Third, parameter penalization and conservative model selection criteria complement the breadth of functional forms spanned by these methods in order to avoid overfit biases.

1.4 What Specific Machine Learning Methods Do We Study?

We select a set of candidate models that are potentially well suited to address the three empirical challenges outlined above. They constitute the canon of methods one would encounter in a graduate level machine learning textbook.[3] This includes linear regression, generalized linear models with penalization, regression trees (including boosted trees and random forests), and neural networks. This is not an exhaustive analysis of all methods as, for example, it excludes some methods that are more commonly employed in problems of classification as opposed to continuous outcomes, such as support vectors. Nonetheless, our list is designed to be representative of predictive analytics tools from various branches of the machine learning toolkit.

1.5 Main Empirical Findings

We conduct a large scale empirical analysis, investigating nearly 30,000 individual stocks over 60 years from 1957 to 2016. Our predictor set includes 95 characteristics for each stock as well as their interaction with 12 aggregate time series variables, totaling to more than 1,300 total baseline signals. Many of the methods we study rapidly proliferate this predictor set further to include non-linear transformations and interactions of the baseline signals. We establish the following empirical facts about machine learning for return prediction.

Machine learning shows great promise for empirical asset pricing. At the broadest level, our main empirical finding is that machine learning as a whole has the potential to improve our empirical understanding of expected asset returns. It metabolizes this set of predictors, which is massive from the perspective of the existing literature, into a return forecasting model that dominates traditional approaches. This in turn means that machine learning can indeed aid in solving practical investments problems such as market timing, portfolio choice, and risk management, justifying its role in the business architecture of the fintech industry.

Consider as a benchmark a panel regression of individual stock returns onto three lagged stock-level characteristics, size, book-to-market, and momentum. This benchmark has a number of attractive features. It is parsimonious and simple. It is also conservative because the characteristics it includes are highly selected (they are routinely demonstrated to be among the most robust re-

[3] See, for example, Hastie et al. (2009).

turn predictors). Lewellen (2015) argues that this model performs about as well as larger and more complex stock prediction models studied in the literature.

In our sample, which is longer and wider than that studied in Lewellen (2015), the out-of-sample R^2 from the benchmark model is 0.17% per month for the panel of individual stock returns. When we expand the OLS panel model to include our set of 1300+ predictors, predictability vanishes immediately—the R^2 drops deeply into negative territory. With so many parameters to estimate, efficiency of traditional regression deteriorates precipitously and therefore produces highly unstable out-of-sample forecasts.

Vast predictor sets are viable for linear prediction when penalization is used. Our first evidence that the machine learning toolkit can benefit return prediction emerges from the fact that straightforward elastic net parameter shrinkage penalization solves the OLS inefficiency problem and pulls the out-of-sample R^2 into positive territory at 0.10% per month. This is in spite of the presence of many likely "fluke" predictors that contribute pure noise to the large model and that must be filtered out. In other words, the high-dimensional predictor set in a simple linear specification is at least competitive with the status quo low-dimensional model, as long as parameterization can be controlled via penalization.

Allowing for non-linearities dramatically improves predictions. Next, we expand the model to accommodate non-linear predictive relationships via generalized additive models, regression trees, and neural networks. We find that trees and neural nets unambiguously improve return prediction with monthly stock-level between 0.24% and 0.39%. But the generalized additive model, which uses spline functions of the baseline predictors along with a group LASSO penalty, fails to robustly outperform the penalized linear specification. This suggests that allowing for (potentially complex) interactions among the baseline predictors is a crucial aspect of non-linearities in the expected return function.[4]

Shallow learning outperforms deeper learning. When we consider a range of neural networks from very shallow (a single hidden layer) to deep networks (up to five hidden layers), we find the shallower networks in general perform best. In the set we consider, two hidden layers almost universally dominate other choices. Likewise, boosted tree and random forest algorithms tend to select trees with few leaves (on average less than six leaves) in our analysis. This is likely an artifact of the relatively small amount of data and tiny signal-to-noise ratio for our return prediction problem, in comparison to the kinds of non-financial settings in which deep learning thrives thanks to astronomically large datasets and strong signals (such as computer vision).

The distance between non-linear methods and the benchmark widens when predicting portfolio returns. We build bottom-up portfolio-level return forecasts from our foundational stock-level model. For example, the monthly R^2 for the S&P 500 portfolio return based on the three-characteristic benchmark model is -0.21%. The generalized additive model exceeds the benchmark in this case at an R^2 of positive 0.21%, while trees and neural nets excel with R^2's from 0.79% to 1.33% per month. This difference is driven by the fact that individual stock return behave erratically for some of the

[4]One of the main differences of the generalized additive model versus trees and neural nets is that our additive model does not consider interactions among the predictors.

smallest and least liquid stocks in our sample. By aggregating to an index of large and liquid firms, we better detect the incremental information gains from machine learning.

The most successful predictors are price trends, liquidity, and beta. All of the methods we study produce a very similar ranking of the most informative stock-level predictors that fall into three main categories. First, and most informative of all, are price trend variables including short-term reversal, stock momentum, and industry momentum. Next are liquidity variables including market value, dollar volume, and bid-ask spread. Finally, market beta and its square are among the leading predictors in all models we consider.

Lastly, we perform a detailed Monte Carlo analysis to help describe the pros and cons of each method under a plausible return data generating process with time-varying betas and dynamic risk premia.

1.6 What Machine Learning Cannot Do

Machine learning has great potential for improving risk premium *measurements*. But, ultimately, these improved predictions are reduced form statistical correlations. The measurements do not tell us about economic *mechanisms* or *equilibria*. There is no sense in which we can rely on machine learning methods to identify deep fundamental associations among asset prices and conditioning variables.

1.7 Literature

Our works extends the empirical literature on stock return prediction, which comes in two basic strands. The first predicts differences in returns across stocks using stock-level characteristics, and is exemplified by Fama and French (2008) and Lewellen (2015). The typical approach in this literature runs cross-sectional returns of future stock returns on a few lagged stock characteristics (or sorts into portfolios on the basis of characteristics—essentially a form of non-parametric regression). The second forecasts the time series of returns and is surveyed by Koijen and Nieuwerburgh (2011) and Rapack and Zhou (2013). This strand of literature typically conducts time series regressions of broad aggregate portfolio returns on a small number macroeconomic predictor variables.

These traditional methods have potentially severe limitations that more advanced statistical tools in machine learning can help overcome. Most important is that regressions and portfolio sorts are ill-suited to handle the large numbers of predictor variables that the literature has accumulated over five decades. The challenge then is how to assess the incremental predictive content of a newly proposed predictor while jointly controlling for the gamut of extant signals (or, relatedly, handling the multiple comparisons problem). Our primary contribution is to demonstrate the potent return predictability harnessable from the large collection of existing variables when machine learning methods are used.

Machine learning methods have appeared sporadically in the asset pricing literature. Several papers apply neural-networks to forecast derivatives prices (Hutchinson et al. (1994) and Yao et al. (2000), among others). Khandani et al. (2010) and Butaru et al. (2016) use regression trees to predict consumer credit card delinquencies and defaults. Heaton et al. (2016) develop a deep learning (neural

network) routine to automate portfolio selection.

Recently, variations of machine learning methods have been used to study the cross section of stock returns. Harvey and Liu (2016) study the multiple comparisons problem using a bootstrap procedure. Giglio and Xiu (2016) and Kelly et al. (2017) use dimension reduction methods to estimate and test factor pricing models. Kozak et al. (2017) and Freyberger et al. (2017) use shrinkage methods to, respectively, approximate a stochastic discount factor and a non-linear function for expected returns. The focus of our paper is to simultaneously explore a wide range of machine learning methods to study the behavior of expected stock returns, with a particular emphasis on the comparative analysis among methods.

2 Methodology

This section describes the collection of machine learning methods that we use in our analysis. In each subsection we introduce a new method and describe it in terms of its three fundamental elements. First is the statistical model describing a method's general functional form for risk premium predictions. The second is an objective function for estimating model parameters. All of our estimates share the basic objective of minimizing mean squared predictions error (MSE). Regularization is introduced through variations on the MSE objective, such as adding parameterization penalties and robustification against outliers. These modifications are designed to avoid problems with overfit and improve models' out-of-sample predictive performance. Finally, even with a small number of predictors, the set of model permutations expands rapidly when one considers non-linear predictor transformations. This proliferation is compounded in our already high dimension predictor set. The third element in each subsection describes computational algorithms for efficiently identifying the optimal specification among the permutations encompassed by a given method.

In our presentation of each method, we aim to provide a sufficiently in depth description of the *statistical model* so that a reader having no machine learning background can understand the basic model structure without needing to consult outside sources. At the same time, when discussing the *computational methods* for estimating each model, we are deliberately terse. There are many variants of each algorithm, and each has its own subtle technical nuances. To avoid bogging the reader down with programming details, we describe of our specific implementation choices in Appendix A and refer readers to original sources for further background.

In its most general form, we describe an asset's excess return as an additive prediction error model:

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

where

$$\mathrm{E}_t(r_{i,t+1}) = g^\star(z_{i,t}). \tag{2}$$

Stocks are indexed as $i = 1, ..., N$ and months by $t = 1, ..., T$. Our objective is to isolate a representation of $\mathrm{E}_t(r_{i,t+1})$ as a function of predictor variables that maximizes the out-of-sample explanatory power for realized $r_{i,t+1}$. We denote those predictors as the P-dimensional vector $z_{i,t}$, and assume

the conditional expected return $g^\star(\cdot)$ is a flexible function of these predictors. Despite its flexibility, this framework imposes some important restrictions. The $g^\star(\cdot)$ function depends neither on i nor t. By maintaining the same form over time and across different stocks, the model leverages information from the entire panel which lends stability to estimates of risk premia for any individual asset. This is in contrast to standard asset pricing approaches that re-estimate a cross-sectional models each time period, or that independently estimate time series models for each stock. Also, $g^\star(\cdot)$ depends on z only through $z_{i,t}$. This means our prediction does not use information from the history prior to t, or from individual stocks other than the ith. That said, our empirical study employs a large number of features motived from theoretical and empirical work in the asset pricing literature. Allowing for extra information is unlikely to improve, and often infeasible given the sample size that are available to us.

2.1 Simple Linear

We begin with the least complex method in our analysis, the simple linear predictive regression model estimated via ordinary least squares (OLS). While we expect this to perform poorly in our high dimension problem, it provides a reference point for emphasizing the distinctive features of more sophisticated methods.

Model. The simple linear model imposes that conditional expectations $g^\star(\cdot)$ can be approximated by a linear function of the raw predictor variables and the parameter vector, θ,

$$g(z_{i,t};\theta) = z_{i,t}'\theta. \tag{3}$$

This model imposes a simple regression specification and does not allow for non-linear effects or interactions of predictors.

Objective Function and Computational Algorithm. Our baseline estimation of the simple linear model uses a standard least squares, or "l_2", objective function:

$$\mathcal{L}(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}\left(r_{i,t+1} - g(z_{i,t};\theta)\right)^2. \tag{4}$$

Minimizing $\mathcal{L}(\theta)$ yields the pooled OLS estimator. The convenience of the baseline l_2 objective function is that it yields analytical estimates and thus avoids sophisticated optimization and computation.

2.1.1 Extension: Robust Objective Functions

In some cases it is possible to improve predictive performance by replacing equation (4) with a weighted least squares objective such as

$$\mathcal{L}_W(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T} w_{i,t}\left(r_{i,t+1} - g(z_{i,t};\theta)\right)^2. \tag{5}$$

This allows the econometrician to tilt estimates towards observations that are more statistically or economically informative. For example, one variation that we consider sets $w_{i,t}$ proportional to the equity market value of stock i at time t. This value-weighted loss function underweights small stocks in favor of large stocks, and is motivated by the economic rational that small stocks represent a large fraction of the traded universe by count while constituting a tiny fraction of aggregate market capitalization.[5]

Heavy tails are a well known attribute of financial returns and stock-level predictor variables. Convexity of the least squares objective (4) places extreme emphasis on large errors, thus outliers can undermine the stability of OLS-based predictions. The statistics literature, long aware of this problem, has developed modified least squares objective functions that tend to produce more stable forecasts than OLS in the presence of extreme observations.[6] In the machine learning literature, a common choice for counteracting the deleterious effect of heavy-tailed observations is the Huber robust objective function, defined as

$$\mathcal{L}_H(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T} H\left(r_{i,t+1} - g(z_{i,t};\theta), \xi\right) \tag{6}$$

where

$$H(x;\xi) = \begin{cases} x^2, & \text{if} \quad |x| \leq \xi; \\ 2\xi|x| - \xi^2, & \text{if} \quad |x| > \xi. \end{cases}$$

The Huber loss, $H(\cdot)$, is a hybrid of squared loss for relatively small errors and absolute loss for relatively large errors, where the combination is controlled by a tuning parameter, ξ, that can be optimized adaptively from the data.[7]

While this detour introduces robust objective functions in the context of the simple linear model, they are indeed easily applicable in almost all of the methods that we study. In our empirical analysis, we study the predictive benefits of robust objective functions, both Huber and market value-weighted, for multiple methods.

2.2 Penalized Linear

Despite its deliberately undemanding form, the simple linear model is likely to fail in the presence of many predictors due to overfit. When the number of predictors P approaches the number of observations T, the linear model becomes inefficient or even inconsistent as it begins to fit noise rather than signal. This is particularly troublesome for the problem of return prediction where the signal-to-noise ratio is notoriously low. We compare P with T instead of $N \times T$ because stock returns maintain a strong cross-sectional dependence induced by a certain factor structure behind

[5] As of Fama and French (2008), the smallest 20% of stocks comprise only 3% of aggregate market capitalization. An example of a statistically motivated weighting scheme uses $w_{i,t}$ inversely proportional to an observation's estimated error variance, a choice that potentially improves prediction efficiency in the spirit of generalized least squares.

[6] Classical analyses include Box (1953), Tukey (1960), and Huber (1964).

[7] OLS is a special case of the (6) with $\xi = \infty$. While most theoretical analysis in high-dimensional statistics assume that data have sub-Gaussian or sub-exponential tails, Fan et al. (2017) provide a theoretical justification of using this loss function in the high-dimensional setting as well as a procedure to determine the tuning parameter.

the prediction errors, so that there is not much independent information from the cross section.

Crucial for avoiding overfit is reducing the number of estimated parameters. The most common machine learning device for imposing parameter parsimony is to append a penalty to the objective function in order to favor more parsimonious specifications. This "regularization" of the estimation problem mechanically deteriorates a model's in-sample performance in hopes that it improves its stability out-of-sample. This will be the case when penalization manages to reduce the model's fit of noise while preserving its fit of the signal.

Objective Function and Computational Algorithm. The statistical model for penalized linear models is the same as the simple linear model in equation (3). That is, it continues to consider only the baseline, untransformed predictors. Instead, penalized methods differ by incorporating a new term in the loss function:

$$\mathcal{L}(\theta;\cdot) = \underbrace{\mathcal{L}(\theta)}_{\text{Loss Function}} + \underbrace{\phi(\theta;\cdot)}_{\text{Penalty}}. \tag{7}$$

There are several choices for the penalty function $\phi(\cdot)$. We focus on the popular "elastic net" penalty, which takes the form

$$\phi(\theta;\lambda,\rho) = \lambda(1-\rho)\sum_{j=1}^{P}|\theta_j| + \frac{1}{2}\lambda\rho\sum_{j=1}^{P}\theta_j^2. \tag{8}$$

The elastic net penalty involves two non-negative tuning parameters λ and ρ, and includes two well known regularizers as special cases. The case ($\rho = 0$) corresponds to the LASSO and uses an absolute value, or "l_1", parameter penalization. The fortunate geometry of the LASSO sets coefficients on a subset of covariates to exactly zero. In this sense, the LASSO imposes sparsity on the specification and can thus be thought of as a variable *selection* method. The case ($\rho = 1$) corresponds to ridge regression, which uses an l_2 parameter penalization, that draws all coefficient estimates closer to zero but does not impose exact zeros anywhere. In this sense, ridge is a *shrinkage* method that helps prevent coefficients from becoming unduly large in magnitude. For intermediate values of ρ, the elastic net encourages simple models through both shrinkage and sparsity.

We adaptively optimize the tuning parameters, λ and ρ, using the validation sample. Our implementation of penalized regression uses the accelerated proximal gradient algorithm and accommodates both least squares and Huber objective functions (see Appendix A.1 for more detail).

2.3 Generalized Additive

Linear models are popular in practice, in part because they can be thought of as a first order approximation to the data generating process. When the "true" model is complex and non-linear, restricting the functional form to be linear introduces approximation error, or model misspecification. Recall that $g^\star(z_{i,t})$ is the true model, suppose $g(z_{i,t};\theta)$ is the functional form specified by the econometrician, and $g(z_{i,t};\widehat{\theta})$ and $\widehat{r}_{i,t+1}$ the fitted model and its ensuing return forecast. We can decompose a

model's forecast error as:

$$r_{i,t+1} - \widehat{r}_{i,t+1} = \underbrace{g^\star(z_{i,t}) - g(z_{i,t};\theta)}_{\text{approximation error}} + \underbrace{g(z_{i,t};\theta) - g(z_{i,t};\widehat{\theta})}_{\text{estimation error}} + \underbrace{\epsilon_{i,t+1}}_{\text{intrinsic error}} .$$

Intrinsic error is irreducible; it is the genuinely unpredictable component of returns associated with news arrival and other sources of randomness in financial markets. Estimation error, which arises due to sampling variation, is determined by the data. It is potentially reducible by adding new observations, though this may not be under the econometrician's control barring from using an efficient procedure. Approximation error is directly controlled by the econometrician, and is reducible by incorporating more flexible specifications that improve the model's ability to approximate the true model. In this and the following subsections, we introduce non-parametric models of $g(\cdot)$ with increasing degrees of flexibility.

Model. The first and most straightforward non-parametric approach that we consider is the generalized additive model. It introduces non-linear transformations of the original predictors as new additive terms in the model. This essentially rolls the transformed variables into an expanded set of predictors in an otherwise linear model. Generalized additive models are thus the closest non-linear counterparts to the linear approaches in Sections 2.1 and 2.2.

The model we study adapts the simple linear form by adding a K-term spline series expansion of the predictors

$$g(z;\theta,p(\cdot)) = \sum_{j=1}^{P} p(z_j)'\theta_j, \tag{9}$$

where the parameters are now a $K \times N$ matrix $\theta = (\theta_1, \theta_2, \ldots, \theta_N)$, and $p(\cdot) = (p_1(\cdot), p_2(\cdot), \ldots, p_K(\cdot))'$ is a vector of basis functions. There are many choices one can choose, among which the splines we adopt are spline series of order 2: $\big(1, z, (z-c_1)^2, (z-c_2)^2, \ldots, (z-c_{K-2})^2\big)$, where c_1, c_2, $\ldots c_{K-2}$ are knots. Higher order splines are allowed for though we do not find them useful in the empirical study.

Objective Function and Computational Algorithm. Because the generalized additive model is an extended linear model, we can use the same set of estimation tools from Section 2.1. In particular, our analysis uses a least squares objective function, both with and without the Huber robustness modification. Because series expansion quickly multiplies the number of model parameters, we use penalization to control degrees of freedom and enhance out-of-sample prediction. We use a penalization function that is specialized for the spline expansion setting and known as the group LASSO. It takes the form

$$\phi(\theta;\lambda,K) = \lambda \sum_{j=1}^{P} \left(\sum_{k=1}^{K} \theta_{j,k}^2 \right)^{1/2} . \tag{10}$$

As its name suggests, the group LASSO includes either all K spline terms associated with a given characteristic or none of them. We embed this penalty in the general objective of equation (7), so group LASSO is usable with either least squares or robust Huber objective, and its accelerated proximal gradient implementation is the same as that for the elastic net (now with tuning parameters,

λ and K).[8]

2.4 Boosted Regression Trees and Random Forests

The model in (9) captures individual predictors' non-linear impact on expected returns, but does not account for interactions among predictors. One way to add interactions is to expand the generalized model to include multivariate functions of predictors. While expanding univariate predictors with K basis functions multiplies the number of parameters by a factor of K, multi-way interactions increase the parameterization combinatorially. Without a priori assumptions for which interactions to include, the generalized additive model becomes computationally infeasible.[9]

Model. As an alternative, regression trees have become a popular machine learning approach for incorporating multi-way predictor interactions into the prediction problem. Unlike linear models, trees are fully nonparametric and possess a logic that departs markedly from traditional regressions. A tree "grows" as a sequence of steps, where at each step a new "branch" sorts the data leftover from the preceding step into two or more bins based on one of the predictor variables. This sequential branching slices the space of predictors into rectangular partitions, and approximates the unknown function $g^\star(\cdot)$ with the average value of the outcome variable within each partition.

Figure 1 shows an example with two predictors, "size" and "b/m." The left panel describes how the tree assigns each observation to a partition based on its predictor values. First, observations are sorted on size. Those above the breakpoint of 0.5 are assigned to Category 3. Those with small size are then further sorted by b/m. Observations with small size and b/m below 0.3 are assigned to Category 1, while those with b/m above 0.3 go into Category 2. Finally, forecasts for observations in each partition are defined as the simple average of the outcome variable's value among observations in that partition.

More formally, the prediction of a tree, $\mathcal{T}$, with K "leaves" (terminal nodes), and depth L, can be written as

$$g(z_{i,t};\theta,K,L)=\sum_{k=1}^{K}\theta_k\mathbf{1}_{\{z_{i,t}\in C_k(L)\}}, \tag{11}$$

where $C_k(L)$ is one of the K partitions of the data. Each partition is a product of up to L indicator functions of the predictors. The constant associated with partition $C_k(L)$, θ_k, is defined as the sample average of outcomes within the partition.[10] In the example of Figure 1, the prediction equation is

$$g(z_{i,t};\theta,3,2)=\theta_1\mathbf{1}_{\{\text{size}_{i,t}<0.5\}}\mathbf{1}_{\{\text{b/m}_{i,t}<0.3\}}+\theta_2\mathbf{1}_{\{\text{size}_{i,t}<0.5\}}\mathbf{1}_{\{\text{b/m}_{i,t}\geq 0.3\}}+\theta_3\mathbf{1}_{\{\text{size}_{i,t}\geq 0.5\}}.$$

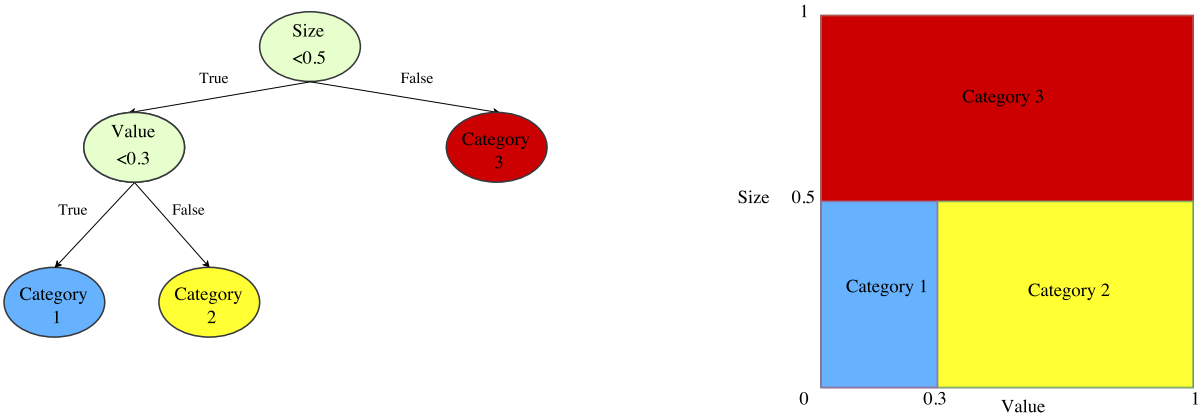
Objective Function and Computational Algorithm. To grow a tree is to find bins that

[8] For additional details, see Appendix A.1.

[9] Parameter penalization does not solve the difficulty of estimating linear models when the number of predictors is exponentially larger than the number of observations. Instead, one must turn to heuristic optimization algorithms such as stepwise regression (sequentially adding/dropping variables until some stopping rule is satisfied), variable screening (retaining predictors whose univariate correlations with the prediction target exceed a certain value), or others.

[10] We focus on recursive binary trees for their relative simplicity. Breiman et al. (1984) discuss more complex tree structures.

Figure 1: Regression Tree Example



Note: This figure presents the diagrams of a regression tree (left) and its equivalent representation (right) in the space of two characteristics (size and value). The terminal nodes of the tree are colored in blue, yellow, and red, respectively. Based on their values of these two characteristics, the entire spectrum of individual stocks are divided into three categories. The tree on the left is applicable to any number of characteristics, whereas it is difficult to use the equivalent representation on the right when there are more than 3 characteristics.

best discriminate among the potential outcomes. The specific predictor variable upon which a branch is based, and the specific value of where the branch is split, is chosen to minimize forecast error. However, the expanse of potential tree structures precludes exact optimization. The literature has developed set of sophisticated optimization heuristics to quickly converge on approximately optimal trees. We follow the algorithm of Breiman et al. (1984), which we describe in detail in Appendix A.2. The basic idea is to myopically optimize forecast error at the start of each branch. At each new level, we choose a sorting variable from the set of predictors and the split value to maximize the discrepancy among average outcomes in each bin.[11] The loss associated with the forecast error for a branch C is often called "impurity." We chooose the most popular l_2 impurity for each branch of the tree:

$$H(\theta, C) = \frac{1}{|C|} \sum_{z_{i,t} \in C} (r_{i,t+1} - \theta)^2, \tag{12}$$

where $|C|$ denotes the number of observations in set C. Given C, it is clear that the optimal choice of θ is: $\theta = \frac{1}{|C|} \sum_{z_{i,t} \in C} r_{i,t+1}$. The procedure is equivalent to finding the branch C that locally minimizes the impurity. Branching halts when the number of leaves or the depth of the tree reach a pre-specified threshold that can be selected adaptively using a validation sample.

Among the advantages of a tree model are that it is invariant to monotonic transformations of predictors, that it naturally accommodates categorical and numerical data in the same model, that it can approximate potentially severe nonlinearities, and that a tree of depth L can capture $(L-1)$-way

[11] Because this is done without consideration of future potential branches, it is possible to bypass a myopically inferior branch that would have led to a future branch with an ultimately superior reduction in forecast error.

interactions. Their flexibility is also their limitation. Trees are among the prediction methods most prone to overfit, and as a result are rarely used without some form of regularization. In our analysis, we consider two "ensemble" tree regularizers that combine forecasts from many different trees into a single forecast.[12]

Boosting. The first regularization method is "boosting," which recursively combines forecasts from many over-simplified trees.[13] Shallow trees on their own are "weak learners" with minuscule predictive power. The theory behind boosting suggests that many weak learners may, as an ensemble, comprise a single "strong learner" with greater stability than a single complex tree.

The details of this boosting procedure are described in Algorithm 3 of Appendix A.2. The resulting estimator is often termed gradient boosted regression tree (GBRT). It starts by fitting a shallow tree (e.g., with depth $L = 1$). This overly simple tree is sure to be a weak predictor with large bias in the training sample. Next, a second simple tree (with the same shallow depth L) is used to fit the prediction residuals from the first tree. Forecasts from these two trees are added together to form an ensemble prediction of the outcome, but the forecast component from the second tree's is shrunken by a factor $\nu \in (0, 1)$ to help prevent the model from overfitting the residuals. At each new step b, a shallow tree is fitted to the residuals of the model with $b - 1$ trees, and its residual forecast is added to the total with a shrinkage weight of ν. This is iterated until there are a total of B trees in the ensemble. The final output is therefore an additive model of shallow trees with three tuning parameters, (L, ν, B), which we adaptively choose in the validation step.

Random Forest. Like boosting, random forest is an ensemble method that combines forecasts from many different trees. It is a variation on a more general procedure known as bootstrap aggregation, or "bagging" (Breiman, 2001). The baseline tree bagging procedure draws B different bootstrap samples of the data, fits a separate regression tree to each, then averages their forecasts. Trees for individual bootstrap samples tend to be deep and overfit, making their individual predictions inefficiently variable. Averaging over multiple predictions reduces this variation, thus stabilizing the trees' predictive performance.

Random forests use a variation on bagging designed to reduce the correlation among trees in different bootstrap samples. If, for example, firm size is the dominant return predictor in the data, then most of the bagged trees will have low-level splits on size resulting in substantial correlation among their ultimate predictions. The forest method "de-correlates" trees by considering only a randomly drawn subset of predictors for splitting at each potential branch. Doing so ensures that, in the example, early branches for at least a few trees will split on characteristics other than firm size. This lowers the average correlation among predictions to further improve the variance reduction relative to standard bagging. Depth L of the trees and number of bootstrap samples B are the tuning parameters optimized via validation. Precise details of our random forest implementation are described in Algorithm 4 of the appendix.

[12] The literature also considers a number of other approaches to tree regularization such as early stopping and post-pruning, both of which are designed to reduce overfit in a single large tree. Ensemble methods demonstrate more reliable performance and are scalable for very large datasets, leading to their increased popularity in recent literature.

[13] Boosting is originally described in Schapire (1990) and Freund (1995) for classification problems to improve the performance of a set of weak learners. Friedman et al. (2000) and Friedman (2001) extend boosting to contexts beyond classification, eventually leading to the gradient boosted regression tree.

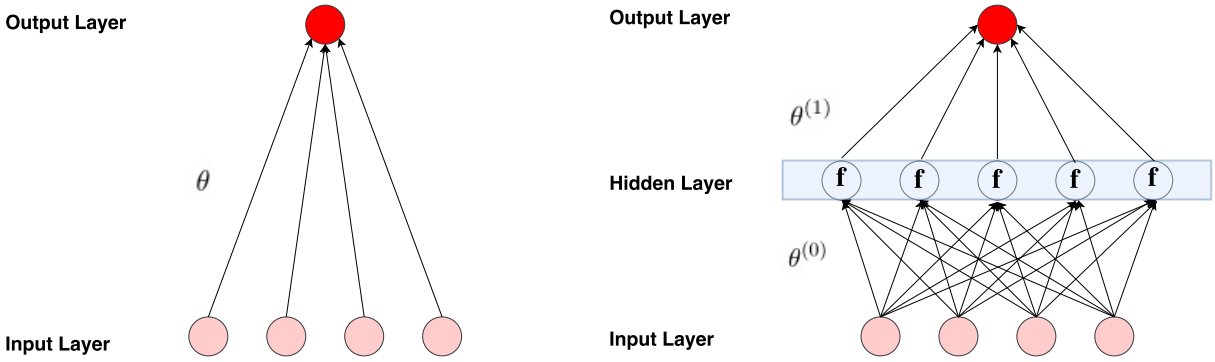
2.5 Neural Networks

The final non-linear method that we analyze is the artificial neural network. Arguably the most powerful modeling device in machine learning, neural networks have theoretical underpinnings as "universal approximators" for any smooth predictive association (Hornik et al., 1989; Cybenko, 1989). They are the currently preferred approach for complex statistical problems such as computer vision, natural language processing, and automated game-playing (such as chess and go). Their flexibility draws from the ability to entwine many telescoping layers of non-linear predictor interactions, earning the synonym "deep learning." At the same time, their complexity ranks neural networks among the least transparent, least interpretable, and most highly parameterized machine learning tools.

Model. We focus our analysis on traditional "feed-forward" networks. These consist of an "input layer" of raw predictors, one or more "hidden layers" that interact and non-linearly transform the predictors, and an "output layer" that aggregates hidden layers into an ultimate outcome prediction. Analogous to axons in a biological brain, layers of the networks represent groups of "neurons" with each layer connected by "synapses" that transmit signals between neurons of different layers. Figure 2 shows two illustrative examples.

The number of units in the input layer is equal to the dimension of the predictors, which we set to four in this example (denoted $z_1, ..., z_4$). The left panel shows the simplest possible network that has no hidden layers. Each of the predictor signals is amplified or attenuated according to a five-dimensional parameter vector, θ, that includes an intercept and one weight parameter per predictor. The output layer aggregates the weighted signals into the forecast $\theta_0 + \sum_{k=1}^{4} z_k\theta_k$. Thus, the simplest neural network is the same as the linear regression model.

Figure 2: Neural Networks



Note: This figure provides diagrams of two simple neural networks without (left) or with one hidden layer (right). The lighter red circles denote the input layer, whereas the darker red circle denotes the output layer. Each arrow is associated with a weight parameter. The right neural network also contains a hidden layer, where a nonlinear activation function f takes action on the aggregated inputs.

The model incorporates more flexible predictive associations by adding hidden layers between the input and output layers. The right panel of Figure 2 shows an example with one hidden layer that

contains five neurons. Each neuron in the hidden layer draws information linearly from all of the input layer units, just as in the simple network on the left. Then, each neuron applies a nonlinear "activation function" f to its aggregated signal before sending its output to the next layer. For example, the second neuron transforms inputs to an output defined as $x_2^{(1)} = f\left(\theta_{2,0}^{(0)} + \sum_{j=1}^{4} z_j \theta_{2,j}^{(0)}\right)$. Lastly, the results from each neuron are linearly aggregated into an output forecast:

$$g(z;\theta) = \theta_0^{(1)} + \sum_{j=1}^{5} x_j^{(1)} \theta_j^{(1)}.$$

Thus, in this example, there are a total of 31 parameters (five parameters to reach each neuron and six weights to aggregate the neurons into a single output).

There are many choices to be made regarding the structure of the neural network, including the number of hidden layers, the number of neurons in each layer, and which units are connected. There are also many choices for the nonlinear activation function (such as sigmoid, hyperbolic, softmax, and tanh). In our implementation, we consider five different architectures, each of which contains 32 neurons allocated throughout the hidden layers. NN1 has one hidden layer with 32 neurons; NN2 has two hidden layers with 16 neurons each; NN3 has three hidden layers with 16, 8, and 8 neurons, respectively; NN4 has four hidden layers with 8 neurons each; and NN5 has five hidden layers with 8, 8, 8, 4, and 4 neurons, respectively. All architectures are fully connected: each unit is connected to all units in the layer below it.[14] By comparing the performance of NN1 through NN5, we can understand the relative benefits of network breadth versus network depth.

We use the same activation function at all nodes, and choose a popular functional form in recent literature known as the rectified linear unit (ReLU). The ReLu function is defined as[15]

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

Our neural network model can be written according to the following general formula. Let $K^{(l)}$ denote the number of neurons in each layer $l = 1, ..., L$. Define the output of neuron k in layer l as $x_k^{(l)}$. Next, define the vector of outputs for this layer (augmented to include a constant, $x_0^{(l)}$) as $x^{(l)} = (1, x_1^{(l)}, ..., x_{K^{(l)}}^{(l)})'$. To initialize the network, similarly define the input layer using the raw predictors, $x^{(0)} = (1, z_1, ..., z_N)'$. The recursive output formula for the neural network at each neuron in layer $l > 0$ is then

$$x_k^{(l)} = \text{ReLU}\left(x^{(l-1)'} \theta_k^{(l-1)}\right), \tag{13}$$

with final output

$$g(z;\theta) = x^{(L-1)'} \theta^{(L-1)}. \tag{14}$$

[14] The corresponding number of weight parameters are 44896, 22720, 22648, 11440, and 11420 for NN1 to NN5, respectively.

[15] See, e.g., Jarrett et al. (2009), Nair and Hinton (2010), and Glorot et al. (2011).

The number of weight parameters in each hidden layer l is $K^{(l)}(1+K^{(l-1)})$, plus another $1+K^{(L-1)}$ weights for the output layer.

Objective Function and Computational Algorithm. We estimate the neural network weight parameters by minimizing the penalized l_2 objective function of prediction errors.[16]

The high degree of non-linearity in neural networks, together with their rich parameterization, make brute force optimization highly computationally intensive (often to the point of infeasibility). A common solution uses stochastic gradient descent (SGD) to train a neural network. Unlike standard descent that uses the entire training sample to evaluate the gradient at each iteration of the optimization, SGD evaluates the gradient from a small random subset of the data. This approximation sacrifices accuracy for enormous acceleration of the optimization routine.

For the same reasons described above (severe nonlinearity and heavy parameterization), regularization of neural networks requires more care than the methods discussed above. We simultaneously employ three regularization techniques in our estimation: learning rate shrinkage, early stopping, and batch normalizations.

A critical tuning parameter in SGD is the learning rate, which controls the step size of the descent. It is necessary to shrink the learning rate toward zero as the gradient approaches zero, otherwise noise in the calculation of the gradient begins to dominate its directional signal. We adopt the "learning rate shrinkage" algorithm of of Kingma and Ba (2014) to adaptively control the learning rate (described further in Algorithm 5 of the Appendix A.3). [17]

Next, "early stopping" is a general machine learning regularization tool. It begins from an initial parameter guess that imposes parsimonious parameterization (for example, setting all θ values to zero). In each step of the optimization algorithm, the parameter guesses are gradually updated to reduce prediction errors in the training sample. At each new guess, predictions are also constructed for the validation sample, and the optimization is terminated when the validation sample errors begin to increase. This typically occurs before the prediction errors are minimized in the training sample, hence the name early stopping (see Algorithm 6). By ending the parameter search early, parameters are shrunken toward the initial guess, and this is how early stopping regularizes against overfit. It is a popular substitute to l_2 penalization of θ parameters because it achieves regularization at a much lower computational cost and automatically determines the appropriate amount of regularization.[18]

"Batch normalization" (Ioffe and Szegedy, 2015) improves optimization by normalizing the inputs of each hidden layer at each iteration of the training (see Algorithm 7). It is effectively an adaptive reparametrization algorithm motivated from the phenomenon of internal covariate shift, i.e., the inputs

[16] Hornik et al. (1989) and White (1989) show that, under reasonable regularity conditions, estimation via l_2 error minimization produces consistent and asymptotically normal forecasts.

[17] Relatedly, random subsetting at each SGD iteration adds noise to the optimization procedure, which itself serves as a form of regularization. See, Wilson and Martinez (2003).

[18] Early stopping bears a comparatively low computation cost because it only partially optimizes, while the l_2-regularization, or more generally elastic net, search across tuning parameters and fully optimizes the model subject to each tuning parameter guess. As usual, elastic net's l_1-penalty component encourages neurons to connect to limited number of other neurons, while its l_2-penalty component shrinks the weight parameters toward zero (a feature known in the neural net literature as "weight-decay"). In certain circumstances, early stopping and weight-decay are shown to be equivalent. See, e.g., Bishop (1995) and Goodfellow et al. (2016). Early stopping can be used alone, or together with l_1-regularization as we do in this paper.

of hidden layers follow different distributions than their counterparts in the validation sample. This issue is constantly encountered when fitting deep neural networks that involve many parameters and rather complex structures. For each hidden unit in each training step, the algorithm normalizes the distribution of of inputs by using the mean and variance of the batch, and introduces a pair of parameters that scale and shift back the normalized value to restore the representation power of the unit. Batch Normalization tends to be a substitute to another popular algorithm "Dropout" (Srivastava et al., 2014), which controls overfit by randomly omitting subsets of predictors at each iteration of the training procedure (analogous to the dropout step in the random forest algorithm). Both share the spirit of regularization by "noising", by multiplying (and subtracting) a random variable to each hidden unit at each step of training.

2.6 Tuning via Validation

The regularization procedures discussed above, which are machine learning's primary defense against overfitting, rely on choices of "tuning" parameters. These tuning parameters are critical to the performance of machine learning methods as they control the model complexity. They include, for example, the penalization parameters λ and ρ in group LASSO and elastic net, the number of iterated trees in boosting, the number of random trees in a forest, and the depth of the trees. In most cases, there is little theoretical guidance for how to choose tuning parameters.

We follow the most common approach in the literature and select tuning parameters adaptively from the data in the validation sample. We divide our sample into three disjoint time periods that maintain the temporal ordering of the data. The first, or "training," subsample is used to estimate the model subject to a specific set of tuning parameter values.

The second, or "validation," sample is used for selecting tuning parameters. We construct forecasts for data points in the validation sample based on the estimated model from the training sample. Next, we calculate the objective function based on forecast errors from the validation sample, and iteratively search for tuning parameters that optimize the validation objective (at each step re-estimating the model from the training data subject to the prevailing tuning parameter values).

Tuning parameters are chosen from the validation sample taking into account estimated parameters, but the parameters estimated from the training data alone. The idea of validation is to simulate an out-of-sample test of the model, and thus searches for a degree of model complexity that tends to produce reliable out-of-sample performance. The validation sample fits are of course not truly out-of-sample because they are used for tuning, which is in turn an input to the estimation. Thus the third, or "testing," subsample, which is used for neither estimation nor tuning, is truly out-of-sample and thus is used to evaluate a method's predictive performance.

We consider a number of sample splitting schemes studied in the forecast evaluation literature (see, e.g., West (2006)). Our baseline analysis follows the "fixed" scheme, and splits the data into training, validation, and testing samples. It estimates the model once from the training and validation samples, and attempts to fit all points in the testing sample using this fixed model estimate.

An alternative is a "rolling" scheme, in which the training and validation samples gradually shift forward in time to include more recent data, but holds the total number of time periods in each train-

ing and validation sample fixed. For each rolling window, one re-fits the model from the prevailing training and validation samples, and tracks a model's performance in the remaining test data that has not been subsumed by the rolling windows. The result is a sequence of performance evaluation measures corresponding to each rolling estimation window. This has the benefit of leveraging more recent information for prediction relative to the fixed scheme.

The third is a "recursive" performance evaluation scheme. Like the rolling approach, it gradually includes more recent observations in the training and validation windows. But the recursive scheme always retains the entire history in the training sample, thus its window size gradually increases. The rolling and recursive schemes are computationally expensive, in particular for more complicated models such as neural networks.

In our empirical exercise, we adopt a hybrid of these schemes, by recursively increasing the training sample, periodically refit the entire model, say, once every two years, and make out-of-sample predictions using the same fitted model over the next two years. Each time we refit, we increase the training sample by two years, while maintaining the same size of rolling sample for validation. We prefer not to cross-validate to maintain the sequence of the data for prediction.

2.7 Performance Evaluation

To assess predictive performance for individual excess stock return forecasts, we calculate the out-of-sample R^2 as

$$R^2_{OOS} = 1 - \frac{\sum_{(i,t)\in\mathcal{T}_3}(r_{i,t+1} - \widehat{r}_{i,t+1})^2}{\sum_{(i,t)\in\mathcal{T}_3} r^2_{i,t+1}}, \tag{15}$$

where $\mathcal{T}_3$ indicates that fits are only assessed on the testing subsample, whose data never enters into model estimation or tuning. This R^2 pools prediction errors across firms and over time into a grand panel-level assessment of each model.

A subtle but important aspect of our R^2 metric is that the denominator is the sum of squared excess returns *without demeaning*. In many out-of-sample forecasting applications, predictions are compared against historical mean returns. While this approach is sensible for the aggregate index, for example, it is flawed when it comes to analyzing individual stock returns. Predicting future excess stock returns with historical averages typically *underperforms* a naive forecast of zero by a large margin. That is, the historical mean stock return is so noisy that it artificially lowers the bar for "good" forecasting performance. We avoid this pitfall by benchmarking our R^2 against a forecast value of zero. To give an indication of the importance of this choice, when we benchmark model predictions against historical mean stock returns, the out-of-sample monthly R^2 of all methods rises by roughly two percentage points.

To make pairwise comparisons of methods, we use the Diebold and Mariano (1995) test for difference in predictive accuracy between two models. This test is well suited for comparing out-of-sample prediction errors, and applies in circumstances where prediction errors are weakly correlated. While time series dependence in returns is sufficiently weak, it is unlikely that the Diebold-Mariano test conditions apply to our stock-level analysis because of potentially strong dependence in the cross

section. We thus use a modified Diebold-Mariano test that compares the cross-sectional average of prediction errors from each model instead of comparing errors among individual returns. More precisely, to test the forecast performance of method (1) versus (2), we define the test statistic $DM_{12} = \bar{d}_{12}/\widehat{\sigma}_{\bar{d}_{12}}$, where

$$d_{12,t+1} = \frac{1}{n_3}\sum_{i=1}^{n_3}\left(\left(\hat{\epsilon}_{i,t+1}^{(1)}\right)^2 - \left(\hat{\epsilon}_{i,t+1}^{(2)}\right)^2\right), \tag{16}$$

$\hat{\epsilon}_{i,t+1}^{(1)}$ and $\hat{\epsilon}_{i,t+1}^{(2)}$ denote the prediction error for stock return i at time t using each method, and n_3 is the number of stocks in the testing sample $\mathcal{T}_3$. Then $\bar{d}_{12}$ and $\widehat{\sigma}_{\bar{d}_{12}}$ denote the mean and Newey-West standard error of $d_{12,t}$ over the testing sample. The Diebold-Mariano test statistic thus compares the average forecast error magnitude of two methods. It is asymptotically normal under mild regularity conditions and thus delivers convenient p-values.

2.8 Interpretation via Variable Importance Plots

The escalation in model complexity renders it increasingly difficult to interpret the results, in particular for the "black-box" models like deep neural networks. Nonetheless, our goal here is rather modest. We aim to identify covariates that have important influences on the cross-section of expected returns. Two caveats are worth emphasizing for interpretation on the selected covariates. First, just like any other return prediction exercise, we do not seek for any causal interpretations beyond suggestive evidence based on variable associations. Secondly, we do not claim the success of identifying the exact set of true factors that determine the cross-section of expected returns, even if the true model is nested within the set of candidate models.

The latter perhaps needs more clarifications. On the one hand, a tension exists in predictive regression settings between the variable selection consistency and the prediction accuracy, regardless of the specific model selection criterions (Information criterions, Regularization, etc), see, e.g., the review article by Ng (2013). The selected model that yields the smallest prediction error tends to include more covariates than those that are present in the true model, whereas asymptotically consistent model selection procedures tend to under-fit the data in finite sample, resulting in larger prediction errors. On the other hand, model selection consistency is not practical. As quoted from Box (1979), all models are wrong, but some are useful. Identifying the covariates in a true model is less realistic as opposed to seeking for a model that yields more accurate predictions. Moreover, even if a true model exists and is part of the candidate sets of models, model selection consistency does not rule out model selection mistakes in any finite sample.

For these reasons, we discover useful covariates through a somewhat ad-hoc approach. For each method, we rank the covariates according to some notion of importance, and regard the covariates that surface among the list of top ones for the majority of methods as useful.

There is not a well-established notion of variable importance (VIP) for most approaches we employ, except for tree-based methods. Breiman et al. (1984) suggest using the average total decrease in

node impurities from splitting on the variable z as its VIP for a single tree $\mathcal{T}$, denoted as $\text{VIP}(z, \mathcal{T})$.[19] For random forest and gradient boosted trees with B trees, we define

$$\text{VIP}(z_j) = \frac{1}{B}\sum_{b=1}^{B}\text{VIP}(z_j, \mathcal{T}_b).$$

For regressions and neural networks, we use the average squared marginal effect for each covariate z_j to characterize its importance:

$$\text{VIP}(z_j) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}\left|\frac{\partial g(z_{i,t};\theta)}{\partial z_j}\right|^2.$$

This importance measure is similar to the absolute marginal effect used in Sirignano et al. (2016) for neural networks. In case of the linear regression, $\text{VIP}(z_j) = \theta_j^2$. Because all covariates are standardized, the proposed measure is comparable in spirit to the MSE impurity for trees.

For convenience in comparison across methods, we standardize these VIPs such that the aggregated total VIP is 1 for each model.

3 An Empirical Study of US Equities

3.1 Data and Overarching Model

We obtain from Center for Research in Security Press (CRSP) monthly total individual equity returns for all firms listed in the NYSE, AMEX, and NASDAQ from March 1957, when S&P 500 first became available, to December 2016, totaling 60 years.[20] The number of stocks in our sample is almost 30,000, with average number of stocks per month exceeding 5,300. We also obtain the Treasury-bill rate as the risk-free rate proxy, with which we calculate individual excess returns.

In addition, we build a large collection stock-level predictive characteristics based on cross section of stock returns literature. These include 95 characteristics (54 of which are updated annually, 14 updated quarterly, and 27 updated monthly). In addition, we include 74 industry dummies corresponding to the first two digits of the Standard Industrial Classification (SIC) codes. We provide the details of these characteristics in Table A.1.[21]

We also construct twelve macroeconomic predictors following the variable definitions detailed in

[19] More specifically, the algorithm to calculate the $\text{VIP}(z, \mathcal{T})$ is given in detail by Algorithm 2 of Appendix A.2.

[20] We include stocks with prices below \$5 or share codes beyond 10 and 11. We do not find it necessary to filter out stocks as the literature typically do when constructing characteristics-sorted portfolios.

[21] The 95 predictive characteristics are based on Green et al. (2013), and we adapt the SAS code available from Jeremiah Green's website and extend the sample period back to 1957. Our data construction differs by adhering more closely to variable definitions in original papers. For example, we construct book-equity and operating profitability following Fama and French (2015). Most of these characteristics are released to the public with a delay. To avoid the forward-looking bias, we assume that monthly characteristics are delayed by at most 1 month, quarterly with at least 4 months lag, and annual with at least 6 months lag. Therefore, in order to predict returns at month $t+1$, we use most recent monthly characteristics at the end of month t, most recent quarterly data by end $t-4$, and most recent annual data by end $t-6$. Another issue is missing characteristics, which we replace with the cross-sectional median at each month for each stock, respectively.

Welch and Goyal (2008), including Dividend Price Ratio (dp), Dividend Yield (dy), Earnings Price Ratio (ep), Dividend Payout Ratio (de), Book-to-Market Ratio (bm), Net Equity Expansion (ntis), Treasury-Bill Rate (tbl), Long Term Yield (lty), Term Spread (tms), Default Spread (dfy), Stock Variance (svar), and one-month-lagged Inflation (infl).[22]

All of the machine learning methods we consider are designed to approximate the overarching empirical model $\mathrm{E}_t(r_{i,t+1}) = g^\star(z_{i,t})$ defined in equation (2). Throughout our analysis we define the baseline set of stock-level covariates $z_{i,t}$ as

$$z_{i,t} = x_t \otimes c_{i,t}, \tag{17}$$

where $c_{i,t}$ is an $P_c \times 1$ matrix of characteristics for each stock i, and x_t is a $P_x \times 1$ vector of macroeconomic predictors for equity risk premia (and are thus common to all stocks, including a constant). Thus, $z_{i,t}$ is an $P \times 1$ vector of features (with $P = P_c P_x$) for predicting individual stock returns, where these predictors are interactions between stock-level characteristics and macroeconomic state variables (including the raw stock characteristics, arising from the interaction of $c_{i,t}$ with the constant element of x_t).

The overarching model specified by (2) and (17) nests many models proposed in the literature (such as Rosenberg (1974) and Harvey and Ferson (1999), among others). One of the leading and motivating examples for this model structure is the standard beta-pricing representation of the asset pricing conditional Euler equation,

$$\mathrm{E}_t(r_{i,t+1}) = \beta_{i,t}' \lambda_t. \tag{18}$$

The structure of our feature set in (17) allows for purely stock-level information to enter expected returns via $c_{i,t}$ in analogy with the risk exposure function $\beta_{i,t}$, and also allows changes in aggregate economic conditions to enter in analogy with the dynamic risk premium λ_t. In particular, if $\beta_{i,t} = \theta_1 c_{i,t}$, and $\lambda_t = \theta_2 x_t$, for some constant parameter matrices θ_1 $(K \times P_c)$ and θ_2 $(K \times P_x)$, then the beta-pricing model in (18) becomes

$$g^\star(z_{i,t}) = \mathrm{E}_t(r_{i,t+1}) = \beta_{i,t}' \lambda_t = c_{i,t}' \theta_1' \theta_2 x_t = (x_t \otimes c_{i,t})' \operatorname{vec}(\theta_1' \theta_2) =: z_{i,t}' \theta,$$

where $\theta = \operatorname{vec}(\theta_1' \theta_2)$. The overarching model is more general than this example because $g^\star(\cdot)$ is not restricted to be a linear function. Considering non-linear $g^\star(\cdot)$ formulations, for example via generalized additive models or neural networks, essentially expands the feature set to include a variety of functional transformations of the baseline $z_{i,t}$ predictor set.

3.2 The Cross-Section of Individual Stocks

As discussed in the previous section, we construct predictors of individual stock returns by combining the aforementioned characteristics panel and time series predictors. The resulting number

[22] The monthly data are available from Amit Goyal's website. We do not use the consumption-wealth ratio in Lettau and Ludvigson (2001) because the data are only available at the quarterly frequency. We do not include either the relative valuation of high- and low-beta stocks in Polk et al. (2006) because the dataset is only available up to December 2002.

Table 1: Comparison of Out-of-Sample Predictive Panel R^2s

	OLS +H	OLS(3) +H	Enet +H	GLasso +H	RF	GBRT +H	NN1	NN2	NN3	NN4	NN5
R^2_{OOS}	-594.70	2.47	2.42	2.26	2.70	2.69	2.47	2.55	2.47	2.47	2.48
R^{2*}_{OOS}	-611.21	0.16	0.10	-0.07	0.39	0.37	0.15	0.24	0.15	0.15	0.17

Note: In this table, we report the out-of-sample (OOS) predictive R^2s for the entire panel of stocks using OLS with size, book-to-market, and momentum, OLS(3), or all variables (OLS), Elastic Net (ENet), Group Lasso (GLasso), Random Forest (RF), Gradient Boosted Regression Tree (GBRT), and five architectures of Neural Networks (NN), respectively. "+H" indicates the use of Huber loss instead of the l_2 loss. We adopt a hybrid scheme for OOS comparison. The first row reports R^2s based on (15), whereas the second row uses the prevailing historical averages for individual stocks as the benchmark for out-of-sample comparison.

of covariates, excluding the constant, is $95 \times (12 + 1) + 74 = 1310$, which include characteristics themselves.

We divide the 60 years of data into 18 years of training sample (1957 - 1974), 12 years of validation sample (1975 - 1986), and the remaining 30 years (1987 - 2016) for out-of-sample testing. Because machine learning algorithms are powerful yet computationally intensive, we do not (need to) recursively refit these models every month. Instead, we only refit every 2 years. To do so, each time we refit, we increase the training sample by 2 years, while maintaining the same size for the validation sample yet rolling it to use the most recent 12 years. Note that we do not use cross-validation in order to maintain the time order of the data. Alternative schemes to divide training and validation may lead to better results, e.g., retraining the model more frequently, but we intend not to data-mine the design of out-of-sample experiments for better results.

We compare in total 11 models, including OLS with all covariates, OLS(3), which preselects size, book-to-market, and momentum as the only covariates, Elastic Net (ENet), Group Lasso (GLasso), Random Forest (RF), Gradient Boosted Regression Tree (GBRT), as well as the aforementioned 5 architectures of Neural Networks (NN1, NN2, ..., NN5).

Table 1 presents the comparison results. For OLS, ENet, GLasso, and GBRT, we only present their robust versions using Huber loss. The winner is RF, followed by GBRT and NNs. NN2 is the best design among all 5 choices. OLS does poorly which may not be surprising, since it is not regularized. There is about 2% inflation in the reported OOS R^2s if the benchmark uses the prevailing historical averages for individual stocks.

Table 2 presents the Diebold-Mariano test statistics for pair-wise comparisons among these models. The results suggest that in terms of panel forecasting, GBRT significantly outperforms RF, which in turn significantly beats NN2, which is moderately the better one among all NNs.

3.3 Market Risk Premia and Relative Returns of Long-Short Portfolios

Next, we examine the predictive performance across all methods in a somewhat more familiar exercise – predicting the market risk premium. We take on a bottom-up strategy to predict portfolio returns

Table 2: Comparison of Out-of-Sample Prediction using Diebold-Mariano Tests

	OLS +H	OLS(3) +H	ENet +H	GLasso +H	RF	GBRT +H	NN1	NN2	NN3	NN4	NN5
OLS+H	-	2.48	2.48	2.48	2.48	2.48	2.48	2.48	2.48	2.48	2.48
OLS(3)+H	-2.96	-	-0.58	-3.21	2.11	2.36	-0.23	1.13	-0.07	-0.02	0.38
ENet+H	-2.96	1.88	-	-1.66	2.15	2.20	0.46	1.22	0.58	0.60	0.79
Glasso+H	-2.96	2.79	0.57	-	3.47	3.80	2.62	3.19	2.63	2.88	3.12
RF	-2.96	-2.66	-2.55	-3.19	-	-0.39	-2.09	-1.33	-2.12	-2.31	-2.06
GBRT+H	-2.97	-4.80	-3.51	-4.16	-2.78	-	-2.04	-1.19	-2.02	-2.24	-2.13
NN1	-2.96	1.13	-1.34	-1.93	3.53	3.90	-	1.94	0.24	0.33	0.81
NN2	-2.96	0.65	-1.56	-2.16	2.91	3.73	-0.56	-	-2.01	-1.75	-1.36
NN3	-2.96	1.45	-1.33	-1.86	3.89	4.37	0.35	1.06	-	0.09	0.64
NN4	-2.96	0.25	-1.78	-2.41	2.84	3.78	-1.43	-0.71	-2.07	-	0.70
NN5	-2.96	0.07	-1.84	-2.43	2.84	4.17	-1.64	-1.04	-2.98	-0.36	-

Note: In this table, we report pair-wise Diebold-Mariano test statistics for the out-of-sample (OOS) panel prediction performance among OLS with size, book-to-market, and momentum, OLS(3), or all variables (OLS), Elastic Net (ENet), Group Lasso (GLasso), Random Forest (RF), Gradient Boosted Regression Tree (GBRT), and five architectures of Neural Networks (NN). "+H" indicates the use of Huber loss instead of the l_2 loss. The upper diagonal uses cross-section mean squared errors as the loss function at each time t, i.e., (16), whereas the lower diagonal uses the mean absolute loss. The null hypothesis is no difference in prediction performance. **Negative numbers indicate the row model outperforms the column model.**

by aggregating individual stock return predictions. Suppose we have a collection of predictions for individual returns next month: $\widehat{r}_{i,t+1}$. Given their weights $w_{i,t}^p$ in the target portfolio p, we can construct a simple predictor for the portfolio return at $t+1$:

$$\widehat{r}_{t+1}^p = \sum_{i=1}^n w_{i,t}^p \times \widehat{r}_{i,t+1}.$$

This bottom-up approach works for any target portfolios once their weights are known a priori. It thereby applies to many portfolios and ETFs for instance.

To demonstrate the empirical relevance of this exercise, following Fama and French (1993) and Fama and French (2015), we create 25 portfolios, including the CRSP value weighted S&P 500 returns, and 4 groups of 3×2 double-sorted portfolios by size and book-to-market, size and investment, size and profitability, and size and momentum.

Table 3 provides the out-of-sample R^2s over the 30-year testing sample. The results are quite encouraging. Among all, the R^2s of NNs are large and mostly positive, all demonstrating substantial gains over the historical means. By the similar calculations of Campbell and Thompson (2008), these R^2s translate to considerable utility gains for asset managers.

It is useful to compare our methodology with those in the existing literature. The majority tests return predictability only using time series of portfolio returns and aggregated variables in a predictive regression setting, e.g., Welch and Goyal (2008), Campbell and Thompson (2008). One exception is Kelly and Pruitt (2013), which design a three-pass filter that leverages the cross-section

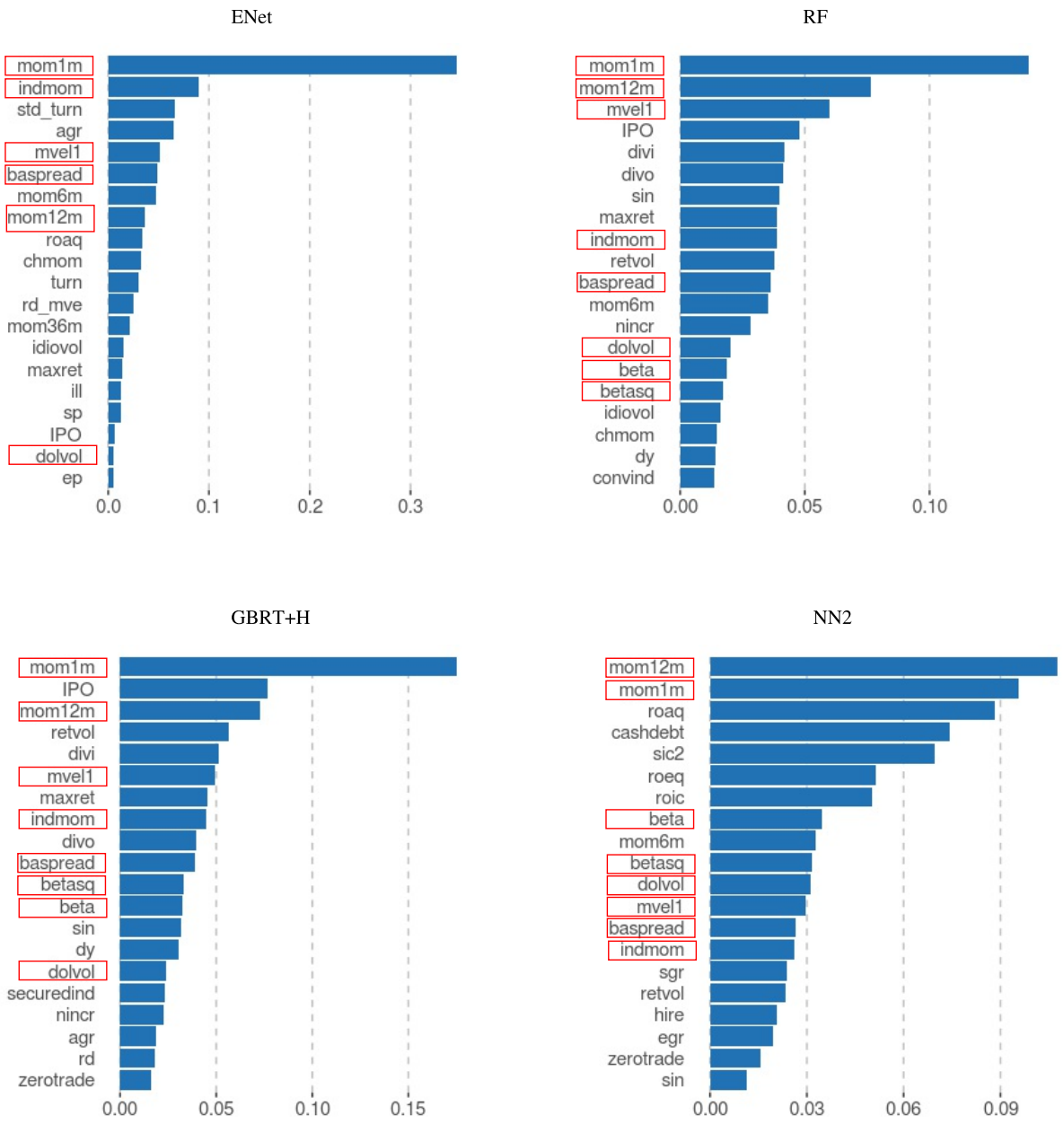
Table 3: Comparison of Out-of-Sample Predictive R^2s for 25 Portfolios

	OLS +H	OLS(3) +H	ENet +H	GLasso +H	RF	GBRT +H	NN1	NN2	NN3	NN4	NN5
VW-S&P500 Index	-785	-0.21	-2.39	0.21	0.79	1.03	1.04	1.33	0.62	0.78	0.59
Big Growth	-1354	0.27	0.07	-0.22	0.84	0.68	0.81	1.34	0.36	0.43	0.58
Big Value	-1538	-1.28	-2.49	0.32	1.04	0.96	1.00	1.24	0.73	0.87	0.64
Big Neutral b/m	-488	0.05	-2.03	-0.04	0.87	0.92	1.17	1.25	0.63	0.77	0.69
Small Growth	-145	0.16	0.81	-0.15	0.64	0.38	0.45	1.12	0.45	0.44	0.64
Small Value	-143	-0.35	-0.97	0.14	0.18	0.35	-0.04	0.12	0.04	0.15	0.31
Small Neutral b/m	-25	-0.37	0.03	0.09	0.35	0.27	0.37	0.99	0.36	0.30	0.55
Big Conservative	-541	-0.33	-2.43	-0.27	0.92	0.66	1.08	1.53	0.65	0.87	0.55
Big Aggressive	-1066	-0.18	-1.56	0.58	1.38	1.16	1.06	1.04	0.65	0.74	0.77
Big Neutral inv	-1592	-0.44	-2.43	-0.04	0.77	0.90	1.05	1.35	0.72	0.85	0.61
Small Conservative	-70	-0.30	-0.09	-0.12	0.62	0.32	0.36	1.04	0.47	0.41	0.60
Small Aggressive	-114	-0.15	-0.54	0.24	0.32	0.40	0.09	0.25	0.08	0.24	0.44
Small Neutral inv	-105	-0.38	0.17	0.08	0.48	0.35	0.39	1.14	0.32	0.31	0.51
Big Robust	-967	-1.78	-3.88	0.00	0.87	0.57	0.87	1.15	0.66	0.75	0.51
Big Neutral op	-344	-0.20	-2.98	0.20	0.69	0.86	0.78	0.81	0.52	0.78	0.53
Big Weak	-339	0.19	-1.29	0.04	0.76	1.03	0.59	0.41	0.30	0.46	0.39
Small Robust	-473	-1.25	-1.05	-0.29	0.38	0.05	0.14	0.49	0.15	0.31	0.46
Small Weak	-344	0.09	-0.49	0.45	0.99	0.66	0.14	0.52	0.37	0.42	0.57
Small Neutral op	-563	-0.62	-1.04	0.03	0.57	0.08	0.13	0.46	0.18	0.25	0.38
Big Up	-1306	-0.47	-2.09	0.07	0.63	0.68	0.92	1.26	0.73	0.98	0.65
Big Down	-632	-0.59	-1.92	-0.35	1.07	1.24	0.80	0.58	0.53	0.55	0.62
Big Medium	-1106	-0.34	-2.93	-0.14	0.90	1.07	1.03	1.11	0.52	0.86	0.59
Small Up	-65	-0.94	-0.11	-0.27	0.02	-0.10	0.31	0.94	0.46	0.25	0.38
Small Down	-102	-0.34	-0.76	0.00	1.60	1.24	0.43	0.60	0.26	0.61	0.74
Small Medium	-331	-0.11	0.27	0.37	0.65	0.70	0.39	0.96	0.19	0.44	0.64

Note: In this table, we report the out-of-sample (OOS) predictive R^2s for 25 portfolios using OLS with size, book-to-market, and momentum, OLS(3), or all variables (OLS), Elastic Net (ENet), Group Lasso (GLasso), Random Forest (RF), Gradient Boosted Regression Tree (GBRT), and five architectures of Neural Networks (NN), respectively. "+H" indicates the use of Huber loss instead of the l_2 loss. The 25 portfolios are 3×2 size double-sorted portfolios used in the construction of five Fama-French factors, including Value (b/m), Investment (inv), Profitability (op), as well as construction of the momentum factor (MOM). We design a hybrid scheme for OOS comparison. We divide the total 60-year sample period into halves. The second half is reserved for OOS testing. We start with the first 30-year sample for training (18 years) and validation (12 years), and refit the model every 2 years. Our training sample increases by 2 years each time we retrain. For validation, we use the latest 12 years prior to prediction for validation.

of disaggregated book-to-market ratios. Our method shares this spirit, yet is built on a simpler approach that aggregates the predicted individual stock returns, which are further constructed using vast amount of information from their characteristics. Our objective function does not align with the prediction of returns of any particular portfolio directly. Rather, we aim at the prediction of the universe of individual stock returns. This strategy is more conservative, which substantially

Figure 3: Variable Importance Plots for Characteristics



Note: This figure presents 4 representative variable importance plots each for top 20 characteristics using, respectively, Elastic Net (ENet), Gradient Boosted Regression Tree (GBRT) plus Huber's loss (+H), Random Forest (RF), and Neural Network (NN2) with 2 hidden layers. Variable importance for all variables associated with each of these characteristics is aggregated. The characteristics highlighted in red boxes are the common ones shared by the 3 nonlinear models.

alleviates the concern of data-snooping. In addition, we adapt a variety of machine learning methods which then serve as robustness checks for one another.

3.4 Which Covariates are Informative about Individual Stock Returns?

Having seen the success of machine learning methods in predicting individual stock returns, we delve into the relative importance of the covariates that drive the results. We present the VIP plots for four models, ENet and GBRT with Huber's loss, RF, and NN2, in Figure 3.

All of the non-linear methods we study produce a very similar ranking of the most informative stock-level predictors that fall into three main categories. First, and most informative of all, are price trend variables including short-term reversal, stock momentum, and industry momentum. Next are liquidity variables including market value, dollar volume, and bid-ask spread. Finally, market beta and its square are among the leading predictors in all models we consider.

3.5 Machine Learning Portfolios

Next, we assess the out-of-sample performance of an investment strategy that forms portfolios on the basis of machine learning predictions. At the end of each month, we calculate the simple average of one-month-ahead out-of-sample stock return predictions from the three top-performing non-linear methods, random forest, gradient boosted regression trees, and the neural network with two hidden layers. We then sort stocks into deciles based on this composite forecast, and reconstitute portfolios each month. Finally, we construct a zero-net-investment portfolio that buys the highest expected return stocks (decile 10) and sells the lowest (decile 1).

Table 4: Performance of Prediction-Sorted Portfolios

	Pred	Avg	Std	SR
Low(L)	0.03	-0.81	5.28	-0.53
2	0.37	0.30	4.22	0.25
3	0.51	0.51	4.04	0.44
4	0.60	0.58	3.82	0.53
5	0.67	0.66	3.76	0.61
6	0.72	0.77	3.75	0.71
7	0.79	0.77	4.03	0.66
8	0.88	0.89	4.45	0.69
9	1.02	1.04	5.35	0.67
High(H)	1.40	2.30	7.24	1.10
H-L	1.37	3.11	4.82	2.23

Note: In this table, we report the performance of prediction-sorted portfolios over the 30-year out-of-sample testing period. All stocks are sorted into deciles based on their predicted returns for the next month. The predictions are based on the simple average of predictions by random forest, gradient boosted regression trees, and neural network with 2 hidden layers. Column "Pred", "Avg", "Std", and "SR" provide the predicted monthly returns for each decile, the average realized monthly returns, their standard deviations, and Sharpe ratios, respectively.

We consider equal-weighted portfolios, which are the most natural portfolios to analyze given that our objective functions minimize an equal weighted average of squared forecast errors. Out-of-

sample portfolio performance aligns exceedingly well machine learning forecasts. Realized returns increase monotonically with machine learning forecasts. And, for all but the most extreme deciles, the quantitative match is between predicted returns and average realized returns is extraordinarily close. The 10–1 portfolio returns on average 3.11% per month. It's monthly volatility is a mere 4.82% (16.7% on an annualized basis), leading to an annualized out-of-sample Sharpe ratio of 2.23.

4 Monte Carlo Simulations

To demonstrate the finite sample performance of all machine learning procedures, we simulate a (latent) 3–factor model for excess returns r_{t+1}, for $t = 1, 2, \ldots, T$:

$$r_{i,t+1} = g^\star(z_{i,t}) + e_{i,t+1}, \quad e_{i,t+1} = \beta_{i,t} v_{t+1} + \varepsilon_{i,t+1}, \quad z_{i,t} = (1, y_t)' \otimes c_{i,t}, \quad \beta_{i,t} = (c_{i1,t}, c_{i2,t}, c_{i3,t}),$$

where c_t is an $N \times m$ matrix of characteristics, v_{t+1} is a 3×1 vector of factors, y_t is a univariate time series, and ε_{t+1} is a $N \times 1$ vector of idiosyncratic errors. We choose $v_{t+1} \sim \mathcal{N}(0, 0.05^2 \times \mathbb{I}_3)$, and $\varepsilon_{i,t+1} \sim t_5(0, 0.05^2)$, in which their variances are calibrated so that the average time series R^2 is 50% and the average annualized volatility is 30%.

We simulate the panel of characteristics for each $1 \leq i \leq N$ and each $1 \leq j \leq 3$ from the following model:

$$c_{ij,t} = \frac{2}{n+1}\text{rank}(\bar{c}_{ij,t}) - 1, \quad \bar{c}_{ij,t} = \rho_j \bar{c}_{ij,t-1} + \epsilon_{ij,t},$$

where $\rho_j \sim \mathcal{U}[0, 1]$, and $\epsilon_{ij,t} \sim \mathcal{N}(0, 1)$, so that the characteristics feature some degree of persistence over time, yet is cross-sectionally normalized to be within $[-1, 1]$. This matches our data cleaning procedure in the empirical study.

In addition, we simulate the time series y_t from the following model:

$$y_t = \rho y_{t-1} + u_t,$$

where $u_t \sim \mathcal{N}(0, 1 - \rho^2)$, and $\rho = 0.9$ so that y_t is highly persistent.

We consider two cases of $g^\star(\cdot)$ functions:

$$\begin{aligned}
&(a) \quad g^\star(z_{i,t}) = (c_{i1,t}, c_{i2,t}, c_{i3,t} \times y_t)\theta_0, \quad \text{where} \quad \theta_0 = (0.02, 0.02, 0.02)';\\
&(b) \quad g^\star(z_{i,t}) = \left(c_{i1,t}^2, c_{i1,t} \times c_{i2,t}, \text{sgn}(c_{i3,t} \times y_t)\right)\theta_0, \quad \text{where} \quad \theta_0 = (0.04, 0.035, 0.01)'.
\end{aligned}$$

In both cases, $g^\star(\cdot)$ only depends on 3 covariates, so there are only 3 non-zero entries in θ, denoted as θ_0. Case (a) is simple and sparse linear model. Case (b) involves a nonlinear covariate $c_{i1,t}^2$, a nonlinear and interaction term $c_{i1,t} \times c_{i2,t}$, and a dummy variable $\text{sgn}(c_{i3,t} \times y_t)$. We calibrate the values of θ_0 such that the cross-sectional R^2 is 25%, and the predictive R^2 is 5%.

Throughout, we fix $N = 200$, $T = 150$, and $P_x = 2$, while comparing the cases of $P_c = 100$ and $m = 50$, corresponding to $P = 200$ and 100, to demonstrate the effect of increasing dimensionality.

For each Monte Carlo sample, we divide the whole time series into 3 consecutive subsamples of equal length for training, validation, and testing, respectively. Specifically, we estimate each of the two models in the training sample, using Ridge, Lasso, Elastic Net (ENet), Group Lasso (GLasso), Random Forest (RF), Gradient Boosted Regression Tree (GBRT), and the same five architectures of Neural Networks (NN) we adopt for the empirical work, respectively, then choose tuning parameters for each method in the validation sample, and calculate the prediction errors in the testing sample. For benchmark, we also compare the pooled OLS with all covariates and that using the oracle model.

Table 5: Comparison of Predictive R^2s for Machine Learning Algorithms in Simulations

Model	(a)				(b)			
Parameter	$P_c = 50$		$P_c = 100$		$P_c = 50$		$P_c = 100$	
$R^2(\%)$	IS	OOS	IS	OOS	IS	OOS	IS	OOS
OLS	7.82	2.04	8.74	0.39	3.44	-2.97	4.41	-4.73
OLS+H	7.59	1.94	8.25	0.20	3.20	-3.11	3.89	-4.99
Lasso	6.26	4.19	6.29	4.20	1.37	0.41	1.38	0.41
Lasso+H	6.17	4.20	6.19	4.21	1.30	0.41	1.30	0.42
Ridge	6.76	3.56	7.06	3.12	1.61	0.18	1.74	0.13
Ridge+H	6.64	3.59	6.89	3.14	1.53	0.18	1.63	0.13
ENet	6.26	4.19	6.29	4.20	1.37	0.41	1.38	0.41
ENet+H	6.17	4.20	6.19	4.21	1.30	0.41	1.31	0.42
Glasso	6.16	3.85	6.17	3.84	3.44	1.02	3.40	0.99
Glasso+H	6.11	3.86	6.10	3.85	3.36	1.04	3.30	1.01
RF	8.40	3.28	8.54	3.26	8.27	2.67	8.49	2.71
GBRT	7.31	3.36	7.32	3.30	6.42	2.61	6.39	2.64
GBRT+H	7.08	3.33	7.17	3.33	6.38	2.72	6.28	2.70
NN1	6.61	3.98	6.67	3.94	5.50	2.05	5.50	1.85
NN2	6.12	3.99	6.22	3.78	5.82	2.11	5.99	1.97
NN3	6.28	3.95	6.32	3.77	5.57	2.02	5.81	1.77
NN4	6.29	3.97	6.35	3.75	5.37	1.97	5.65	1.73
NN5	6.32	3.87	6.47	3.70	5.20	1.77	4.78	1.20
Oracle	6.25	5.06	6.25	5.06	5.55	5.12	5.55	5.12

Note: In this table, we report the average in-sample (IS) and out-of-sample (OOS) R^2s for models (a) and (b) using Ridge, Lasso, Elastic Net (ENet), Group Lasso (GLasso), Random Forest (RF), Gradient Boosted Regression Tree (GBRT), and five architectures of Neural Networks (NN), respectively. "+H" indicates the use of Huber loss instead of the l_2 loss. "Oracle" stands for using the true covariates in a pooled-OLS regression. We fix $N = 200$, $T = 150$, and $P_x = 2$, comparing $P_c = 100$ with $P_c = 50$. The number of Monte Carlo repetitions is 100.

We report the average R^2s both in-sample (IS) and out-of-sample (OOS) for each model and each method over 100 Monte Carlo repetitions in Table 5. Both IS and OOS R^2s are relative to the estimator based on the IS average. For model (a), Lasso and ENet deliver the best OOS R^2. This is not surprising given that the true model is sparse and linear in the input covariates. More advanced methods such as RF, NN, and GBRT tend to overfit, so their performance is slightly worse. By contrast, for model (b), these methods clearly dominate Lasso and ENet, because the latter cannot capture the nonlinearity in the model. GLasso is slightly better, but is dominated by NNs, RF, and

GBRT. OLS is the worst in all settings, not surprisingly. Also, using Huber loss improves the OOS performance. When m increases, IS R^2s increase whereas OOS R^2s decrease. Hence, the performance of all methods deteriorates as overfitting exacerbates. GBRT plus Huber loss remain the best choice for the nonlinear model.

Table 6: Comparison of Average Variable Selection Frequencies in Simulations

Model (a)

Parameter	Method	$c_{i1,t}$	$c_{i2,t}$	$c_{i3,t}$	$c_{i1,t} \times y_t$	$c_{i2,t} \times y_t$	$c_{i3,t} \times y_t$	Noise
$P_c = 50$	Lasso	0.96	0.95	0.60	0.57	0.60	0.92	0.08
	Lasso+H	0.96	0.95	0.59	0.53	0.57	0.92	0.08
	ENet	0.96	0.95	0.60	0.57	0.60	0.92	0.08
	ENet+H	0.96	0.95	0.59	0.53	0.57	0.92	0.08
	GLasso	0.95	0.97	0.62	0.65	0.68	0.94	0.13
	GLasso+H	0.95	0.97	0.62	0.67	0.65	0.94	0.12
$P_c = 100$	Lasso	0.96	0.95	0.59	0.55	0.58	0.92	0.06
	Lasso+H	0.96	0.95	0.57	0.52	0.55	0.92	0.05
	ENet	0.96	0.95	0.59	0.55	0.58	0.92	0.06
	ENet+H	0.96	0.95	0.57	0.54	0.55	0.92	0.05
	GLasso	0.95	0.97	0.60	0.61	0.61	0.94	0.09
	GLasso+H	0.95	0.97	0.59	0.63	0.60	0.94	0.09

Model (b)

Parameter	Method	$c_{i1,t}$	$c_{i2,t}$	$c_{i3,t}$	$c_{i1,t} \times y_t$	$c_{i2,t} \times y_t$	$c_{i3,t} \times y_t$	Noise
$P_c = 50$	Lasso	0.36	0.25	0.32	0.36	0.32	0.72	0.03
	Lasso+H	0.33	0.26	0.27	0.31	0.28	0.72	0.03
	ENet	0.36	0.25	0.33	0.36	0.32	0.72	0.03
	ENet+H	0.33	0.26	0.27	0.30	0.28	0.72	0.03
	GLasso	0.87	0.57	0.65	0.74	0.67	0.82	0.19
	GLasso+H	0.87	0.56	0.62	0.74	0.67	0.82	0.19
$P_c = 100$	Lasso	0.33	0.25	0.29	0.33	0.31	0.72	0.02
	Lasso+H	0.30	0.25	0.25	0.28	0.26	0.72	0.02
	ENet	0.33	0.25	0.30	0.33	0.31	0.72	0.02
	ENet+H	0.30	0.25	0.25	0.28	0.26	0.72	0.02
	GLasso	0.87	0.52	0.60	0.71	0.64	0.81	0.12
	GLasso+H	0.87	0.50	0.56	0.72	0.64	0.82	0.12

Note: In this table, we report the average variable selection frequencies of 6 particular covariates for models (a) and (b) using Lasso, Elastic Net (ENet), and Group Lasso (GLasso), respectively. "+H" indicates the use of Huber loss instead of the l_2 loss. Column "Noise" reports the average selection frequency of the remaining $2m - 6$ covariates. We fix $N = 200$, $T = 150$, and $P_x = 2$, comparing $P_c = 100$ with $P_c = 50$. The number of Monte Carlo repetitions is 100.

We also report the average variable selection frequencies of 6 particular covariates and the average of the remaining $2m - 6$ covariates for models (a) and (b) in Table 6, using Lasso, Elastic Net, and Group Lasso and their robust versions. We focus on these methods because they all impose the l_1 penalty and hence encourage variable selection. As expected, for model (a), the true covariates

($c_{i1,t}$, $c_{i2,t}$, $c_{i3,t} \times y_t$) are selected in over 90% of the sample paths, whereas correlated yet redundant covariates ($c_{i3,t}$, $c_{i1,t} \times y_t$, $c_{i2,t} \times y_t$) are also selected in around 60% of the samples. By contrast, the remaining covariates are rarely selected. Although model selection mistakes are unavoidable, perhaps due to the tension between variable selection and prediction or for finite sample issues, the true covariates are part of the selected models with high probabilities. For model (b), while no covariates are part of the true model, the 6 covariates we present are more relevant, and hence selected substantially more frequently than the remaining $2m - 6$ ones.

Finally, we report the average VIPs of the 6 particular covariates and the average of the remaining $2m-6$ covariates for models (a) and (b) in Table 7, using Random Forest (RF) and Gradient Boosted Regression Tree (GBRT), along with Neural Networks. We find similar results for both models (a) and (b) that the 6 covariates we present are substantially more important than the remaining $2m - 6$ ones. NNs tend to overestimate the importance of noise variables, perhaps because of its overwhelmingly large number of parameters related to the noise variables, which then explains their slightly worse performance against GBRT in the OOS comparison.

Overall, the simulation results suggest that the machine learning methods are successful in singling out informative variables, even though highly correlated covariates are difficult to distinguish. This is not surprising, as these methods are implemented to improve prediction, for which purpose the best model often does not agree with the true model, in particular when covariates are highly correlated.

5 Conclusion

Using the empirical context of return prediction as a proving ground, we perform a comparative analysis of methods that constitute the machine learning repertoire. At the highest level, our findings demonstrate great promise for machine learning methods to improve our empirical understanding of asset prices. The best performing methods are regression trees and neural networks, and we isolate the source of their predictive advantage to accommodation of non-linear interactions between the baseline predictors that is missed by other methods. We also find that "shallow" learning outperforms "deep" learning, which differs from the typical conclusion in other fields such as computer vision or bioinformatics, and is undoubtedly due the comparative dearth of data in asset pricing problems. We also find that machine learning methods are most valuable for forecasting larger and more liquid stock returns, and this can be traced to the comparatively low signal to noise ratios in small, illiquid stock price fluctuations. Lastly, we find that *all* methods agree on a fairly small set of dominant predictive signals that fall into three distinct groups. The most powerful predictors are typically associated with technical analysis and include return reversal and momentum. The next most powerful predictors are measures of stock liquidity, including market capitalization, volume, and bid-ask spreads. The final group summarizes exposure to aggregate market fluctuations, and includes market beta and its square.

The overall success of machine learning algorithms for return prediction brings promise for both economic modeling and for practical aspects of the financial industry. By better measuring risk

Table 7: Comparison of Average Variable Importance in Simulations

Model (a)

Parameter	Method	$c_{i1,t}$	$c_{i2,t}$	$c_{i3,t}$	$c_{i1,t} \times y_t$	$c_{i2,t} \times y_t$	$c_{i3,t} \times y_t$	Noise
$P_c = 50$	RF	20.47	19.92	3.72	6.30	5.79	20.57	0.25
	GBRT	22.48	21.37	4.16	7.79	7.18	26.89	0.11
	GBRT+H	23.02	22.02	4.02	7.77	6.89	28.16	0.09
	NN1	20.47	20.75	3.02	3.09	3.23	16.68	0.35
	NN2	15.92	14.74	2.78	2.40	2.30	12.28	0.53
	NN3	5.75	5.25	2.21	1.88	1.76	4.01	0.84
	NN4	3.29	2.57	1.63	1.56	1.55	2.26	0.93
	NN5	2.25	1.87	1.66	1.62	1.68	2.09	0.95
$P_c = 100$	RF	20.01	19.44	3.62	5.57	5.37	19.93	0.13
	GBRT	22.16	20.74	3.90	7.44	7.26	26.76	0.06
	GBRT+H	22.82	21.89	3.88	7.25	6.87	27.06	0.05
	NN1	15.86	15.79	2.61	2.46	2.43	13.67	0.24
	NN2	9.74	9.26	1.60	1.66	1.21	7.33	0.36
	NN3	4.34	2.79	1.00	1.05	0.84	3.52	0.45
	NN4	1.18	1.83	0.86	0.63	0.79	1.32	0.48
	NN5	1.24	1.23	0.86	0.61	0.60	0.96	0.49

Model (b)

Parameter	Method	$c_{i1,t}$	$c_{i2,t}$	$c_{i3,t}$	$c_{i1,t} \times y_t$	$c_{i2,t} \times y_t$	$c_{i3,t} \times y_t$	Noise
$P_c = 50$	RF	21.27	5.99	3.26	9.14	5.10	28.43	0.29
	GBRT	29.56	8.98	4.58	12.43	8.39	27.13	0.09
	GBRT+H	29.64	9.60	4.56	12.41	8.43	26.94	0.09
	NN1	6.06	4.69	6.09	5.53	4.41	18.43	0.58
	NN2	4.08	2.72	3.87	2.83	2.49	8.31	0.81
	NN3	2.08	1.93	2.45	2.28	1.56	3.42	0.92
	NN4	1.34	1.36	1.72	1.32	1.49	2.88	0.96
	NN5	2.06	1.66	2.74	2.03	1.55	4.15	0.91
$P_c = 100$	RF	19.61	5.01	3.23	8.30	4.33	27.01	0.17
	GBRT	29.03	9.22	4.59	12.04	7.78	26.28	0.06
	GBRT+H	30.30	10.00	4.47	11.66	7.92	26.73	0.05
	NN1	3.66	3.00	4.23	3.79	3.55	12.75	0.36
	NN2	1.68	1.24	2.22	1.45	1.24	4.27	0.45
	NN3	1.07	0.60	1.02	0.60	0.67	1.54	0.49
	NN4	0.55	0.46	0.57	0.60	0.66	0.69	0.50
	NN5	0.91	0.95	1.88	0.81	0.61	1.53	0.48

Note: In this table, we report the average variable importance of 6 particular covariates for models (a) and (b) using Random Forest (RF), Gradient Boosted Regression Tree (GBRT), and five architectures of Neural Networks (NN), respectively. "+H" indicates the use of Huber loss instead of the l_2 loss. Column "Noise" reports the average variable importance of the remaining $2m - 6$ covariates. We fix $N = 200$, $T = 150$, and $P_x = 2$, comparing $P_c = 100$ with $P_c = 50$. The number of Monte Carlo repetitions is 100.

premia through machine learning, the challenge of identifying reliable economic mechanisms to asset pricing phenomenon becomes slightly becomes less steep. And the consistency of our findings across methods, and for individual stocks and portfolios alike, helps justify the growing role of machine learning throughout the architecture of the burgeoning fintech industry.

References

Bishop, Christopher M., 1995, *Neural Networks for Pattern Recognition* (Oxford University Press).

Box, G. E. P., 1953, Non-normality and tests on variances, *Biometrika* 40, 318–335.

Box, G. E. P., 1979, Robustness in the strategy of scientific model building, in R. L. Launer, and G. N. Wilkinson, eds., *Robustness in Statistics* (Academic Press).

Breiman, Leo, 2001, Random forests, *Machine learning* 45, 5–32.

Breiman, Leo, Jerome Friedman, Charles J Stone, and Richard A Olshen, 1984, *Classification and regression trees* (CRC press).

Bühlmann, Peter, and Torsten Hothorn, 2007, Boosting Algorithms: Regularization, Prediction and Model Fitting, *Statistical Science* 22, 477–505.

Butaru, Florentin, Qingqing Chen, Brian Clark, Sanmay Das, Andrew W Lo, and Akhtar Siddique, 2016, Risk and risk management in the credit card industry, *Journal of Banking & Finance* 72, 218–239.

Campbell, John Y., and S. B. Thompson, 2008, Predicting excess stock returns out of sample: Can anyting beat the historical average?, *Review of Financial Studies* 21, 1509–1531.

Cybenko, George, 1989, Approximation by superpositions of a sigmoidal function, *Mathematics of control, signals and systems* 2, 303–314.

Daubechies, I, M Defrise, and C De Mol, 2004, An iterative thresholding algorithm for linear inverse problems with a sparsity constraint, *Communications on Pure and Applied Mathematics* 57, 1413–1457.

Diebold, Francis X., and Roberto S. Mariano, 1995, Comparing predictive accuracy, *Journal of Business & Economic Statistics* 13, 134–144.

Fama, Eugene F, and Kenneth R French, 1993, Common risk factors in the returns on stocks and bonds, *Journal of financial economics* 33, 3–56.

Fama, Eugene F, and Kenneth R French, 2008, Dissecting anomalies, *The Journal of Finance* 63, 1653–1678.

Fama, Eugene F., and Kenneth R. French, 2015, A five-factor asset pricing model, *Journal of Financial Economics* 116, 1–22.

Fan, Jianqing, Quefeng Li, and Yuyan Wang, 2017, Estimation of high dimensional mean regression in the absence of symmetry and light tail assumptions, *Journal of the Royal Statistical Society, B* 79, 247–265.

Freund, Yoav, 1995, Boosting a weak learning algorithm by majority, *Information and Computation* 121, 256–285.

Freyberger, Joachim, Andreas Neuhierl, and Michael Weber, 2017, Dissecting characteristics nonparametrically, Technical report, University of Wisconsin-Madison.

Friedman, Jerome, Trevor Hastie, Holger Höfling, Robert Tibshirani, et al., 2007, Pathwise coordinate optimization, *The Annals of Applied Statistics* 1, 302–332.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani, 2000, Additive logistic regression: A statistical view of boosting, *The Annals of Statistics* 28, 337–374.

Friedman, Jerome H, 2001, Greedy function approximation: a gradient boosting machine, *Annals of statistics* 1189–1232.

Giglio, Stefano W, and Dacheng Xiu, 2016, Inference on risk premia in the presence of omitted factors, Technical report, University of Chicago.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio, 2011, Deep sparse rectifier neural networks., in *Aistats*, volume 15, 315–323.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville, 2016, *Deep Learning* (MIT Press), http://www.deeplearningbook.org.

Green, Jeremiah, John RM Hand, and X Frank Zhang, 2013, The supraview of return predictive signals, *Review of Accounting Studies* 18, 692–730.

Harvey, Campbell R., and Wayne E. Ferson, 1999, Conditioning variables and the cross-section of stock returns, *Journal of Finance* 54, 1325–1360.

Harvey, Campbell R, and Yan Liu, 2016, Lucky factors, Technical report, Duke University.

Harvey, Campbell R, Yan Liu, and Heqing Zhu, 2016, ... and the cross-section of expected returns, *Review of Financial Studies* 29, 5–68.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman, 2009, *The Elements of Statistical Learning* (Springer).

Heaton, JB, NG Polson, and JH Witte, 2016, Deep learning in finance, *arXiv preprint arXiv:1602.06561* .

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White, 1989, Multilayer feedforward networks are universal approximators, *Neural networks* 2, 359–366.

Huber, P. J., 1964, Robust estimation of a location parameter, *Annals of Mathematical Statistics* 35, 73–101.

Hutchinson, James M, Andrew W Lo, and Tomaso Poggio, 1994, A nonparametric approach to pricing and hedging derivative securities via learning networks, *The Journal of Finance* 49, 851–889.

Ioffe, Sergey, and Christian Szegedy, 2015, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *International Conference on Machine Learning* 448–456.

Jarrett, Kevin, Koray Kavukcuoglu, Yann Lecun, et al., 2009, What is the best multi-stage architecture for object recognition?, in *2009 IEEE 12th International Conference on Computer Vision*, 2146–2153, IEEE.

Kelly, Bryan, and Seth Pruitt, 2013, Market expectations in the cross-section of present values, *The Journal of Finance* 68, 1721–1756.

Kelly, Bryan, Seth Pruitt, and Yinan Su, 2017, Some characteristics are risk exposures, and the rest are irrelevant, Technical report, University of Chicago.

Khandani, Amir E, Adlar J Kim, and Andrew W Lo, 2010, Consumer credit-risk models via machine-learning algorithms, *Journal of Banking & Finance* 34, 2767–2787.

Kingma, Diederik, and Jimmy Ba, 2014, Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980* .

Koijen, Ralph, and Stijn Van Nieuwerburgh, 2011, Predictability of returns and cash flows, *Annual Review of Financial Economics* 3, 467–491.

Kozak, Serhiy, Stefan Nagel, and Shrihari Santosh, 2017, Shrinking the cross section, Technical report, University of Michigan.

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

Lewellen, Jonathan, 2015, The cross-section of expected stock returns, *Critical Finance Review* 4, 1–44.

Nair, Vinod, and Geoffrey E Hinton, 2010, Rectified linear units improve restricted boltzmann machines, in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 807–814.

Nesterov, Yurii, 1983, A method of solving a convex programming problem with convergence rate $o(1/k^2)$, *Soviet Mathematics Doklady* 27, 372–376.

Ng, Serena, 2013, Variable selection in predictive regressions, in Graham Elliott, and Allan Timmermann, eds., *Handbook of Economic Forecasting*, volume 2B, 752–789 (Elsevier).

Parikh, Neal, and Stephen Boyd, 2013, Proximal algorithms, *Foundations and Trends in Optimization* 1, 123–231.

Polk, C., S. Thompson, and T. Vuolteenaho, 2006, Cross-section forecasts of the equity premium, *Journal of Financial Economics* 81, 101–141.

Polson, Nicholas G., James Scott, and Brandon T. Willard, 2015, Proximal algorithms in statistics and machine learning, *Statistical Science* 30, 559–581.

Rapack, David, and Guofu Zhou, 2013, Forecasting stock returns, in Graham Elliott, and Allan Timmermann, eds., *Handbook of Economic Forecasting*, volume 2A, 328–383 (Elsevier).

Rosenberg, Barr, 1974, Extra-market components of covariance in security returns, *Journal of Financial and Quantitative Analysis* 9, 263–274.

Schapire, Robert E., 1990, The strength of weak learnability, *Machine Learning* 5, 197–227.

Sirignano, Justin, Apaar Sadhwani, and Kay Giesecke, 2016, Deep learning for mortgage risk, *Available at SSRN 2799443* .

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, 2014, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *The Journal of Machine Learning Research* 15, 1929–1958.

Tukey, J. W., 1960, A survey of sampling from contaminated distributions, in I. Olkin, S. G. Ghurye, W. Hoeffding, W. G. Madow, and H. B. Mann, eds., *Contributions to probability and statistics: Essays in Honor of Harold Hotelling* (Stanford University Press).

Welch, Ivo, and Amit Goyal, 2008, A Comprehensive Look at The Empirical Performance of Equity Premium Prediction, *Review of Financial Studies* 21, 1455–1508.

West, Kenneth D., 2006, Forecast evaluation, in Graham Elliott, Cliver Granger, and Allan Timmermann, eds., *Handbook of Economic Forecasting*, volume 1, 99–134 (Elsevier).

White, Halbert, 1989, Learning in artificial neural networks: A statistical perspective, *Neural computation* 1, 425–464.

Wilson, D. Randall, and Tony R. Martinez, 2003, The general inefficiency of batch training for gradient descent learning, *Neural networks* 16, 1429–1451.

Yao, Jingtao, Yili Li, and Chew Lim Tan, 2000, Option price forecasting using neural networks, *Omega* 28, 455–466.

A Algorithms in Details

A.1 Lasso, Ridge, Elastic Net, and Group Lasso

We present the accelerated proximal algorithm (APG), see, e.g., Parikh and Boyd (2013) and Polson et al. (2015)., which allows for efficient implementation of the Elastic Net, Lasso, Ridge regression, and Group Lasso for both l_2 and Huber losses. We rewrite their regularized objective functions as

$$\mathcal{L}(\theta;\cdot) = \underbrace{\mathcal{L}(\theta)}_{\text{Loss Function}} + \underbrace{\phi(\theta;\cdot)}_{\text{Penalty}}, \tag{A.1}$$

where we omit the dependence on the tuning parameters. Specifically, we have

$$\phi(\theta;\cdot) = \begin{cases} \frac{1}{2}\lambda\sum_{j=1}^{P}\theta_j^2, & \text{Ridge;} \\ \lambda\sum_{j=1}^{P}|\theta_j|, & \text{Lasso;} \\ \lambda(1-\rho)\sum_{j=1}^{P}|\theta_j| + \frac{1}{2}\lambda\rho\sum_{j=1}^{P}\theta_j^2, & \text{Elastic Net;} \\ \lambda\sum_{j=1}^{P}\|\theta_j\|, & \text{Group Lasso.} \end{cases}, \tag{A.2}$$

where in the Group Lasso case, $\theta = (\theta_1, \theta_2, \ldots, \theta_P)$ is a $K \times P$ matrix.

Proximal algorithms are a class of algorithms for solving convex optimization problems, in which the base operation is evaluating the proximal operator of a function, ie., solving a small convex optimization problem. In many cases, this smaller problem has a closed form solution. The proximal operator is defined as:

$$\mathbf{prox}_{\gamma f}(\theta) = \underset{z}{\operatorname{argmin}}\left\{ f(z) + \frac{1}{2\gamma}\|z-\theta\|^2 \right\}.$$

An important property of the proximal operator is that the minimizer of a convex function $f(\cdot)$ is a fixed point of $\mathbf{prox}_f(\cdot)$, i.e., $\theta^\star$ minimizes $f(\cdot)$ if and only if

$$\theta^\star = \mathbf{prox}_f(\theta^\star).$$

The proximal gradient algorithm is designed to minimize an objective function of the form (A.1), where $\mathcal{L}(\theta)$ is differentiable function of θ but $\phi(\theta;\cdot)$ is not. Using properties of the proximal operator, one can show that $\theta^\star$ minimizes (A.1), if and only if

$$\theta^\star = \mathbf{prox}_{\gamma\phi}(\theta^\star - \gamma\nabla\mathcal{L}(\theta^\star)).$$

This result motivates the first two iteration steps in Algorithm 1. The third step inside the while

loop is a Nesterov momentum (Nesterov (1983)) adjustment that accelerates convergence.

The optimization problem requires the proximal operators of $\phi(\theta;\cdot)$s in (A.2), which have closed forms:

$$\mathbf{prox}_{\gamma\phi}(\theta) = \begin{cases} \dfrac{\theta}{1+\lambda\gamma}, & \text{Ridge;} \\ \lambda S(\theta, \lambda\gamma), & \text{Lasso;} \\ \dfrac{1}{1+\lambda\gamma\rho} S(\theta, (1-\rho)\lambda\gamma), & \text{Elastic Net;} \\ (\widetilde{S}(\theta_1, \lambda\gamma)^\intercal, \widetilde{S}(\theta_2, \lambda\gamma)^\intercal, \ldots, \widetilde{S}(\theta_P, \lambda\gamma)^\intercal)^\intercal, & \text{Group Lasso.} \end{cases},$$

where $S(x,\mu)$ and $\widetilde{S}(x,\mu)$ are vector-valued functions, whose ith components are defined by:

$$(S(x,\mu))_i = \begin{cases} x_i - \mu, & \text{if } x_i > 0 \text{ and } \mu < |x_i|; \\ x_i + \mu, & \text{if } x_i < 0 \text{ and } \mu < |x_i|; \\ 0, & \text{if } \mu \geq |x_i|. \end{cases}, \quad (\widetilde{S}(x,\mu))_i = \begin{cases} x_i - \mu\frac{x_i}{\|x_i\|}, & \text{if } \|x_i\| > \mu; \\ 0, & \text{if } \|x_i\| \leq \mu. \end{cases}.$$

Note that $S(x,\mu)$ is the soft-thresholding operator, so the proximal algorithm is equivalent to the coordinate descent algorithm in the case of l_2 loss, see, e.g., Daubechies et al. (2004), Friedman et al. (2007). The proximal framework we adopt here allows efficient implementation of Huber loss and convergence acceleration.

Algorithm 1: Accelerated Proximal Gradient Method

Initialization: $\theta_0 = 0$, $m = 0$, γ;
while θ_m *not converged* **do**
  $\bar{\theta} \leftarrow \theta_m - \gamma\nabla\mathcal{L}(\theta)\,|_{\theta=\theta_m}$.
  $\tilde{\theta} \leftarrow \mathbf{prox}_{\gamma\phi}(\bar{\theta})$.
  $\theta_{m+1} \leftarrow \tilde{\theta} + \frac{m}{m+3}(\tilde{\theta} - \theta_m)$.
  $m \leftarrow m + 1$.
end
Result: The final parameter estimate is θ_m.

A.2 Tree, Random Forest, and Gradient Boosted Tree

Algorithm 2 is a greedy algorithm, see, e.g., Breiman et al. (1984), to grow a complete binary regression tree. Next, Algorithm 4 yields the random forest, e.g., Hastie et al. (2009). Finally, Algorithm 3 delivers the gradient boosted tree (Friedman (2001)), for which we follow the version written by Bühlmann and Hothorn (2007).

Algorithm 2: Classification and Regression Tree

Initialize the stump. $C_1(0)$ denotes the range of all covariates, $C_l(d)$ denote the l-th node of depth d.
for d *from* 1 *to* L **do**
  for i *in* $\{C_l(d-1), l = 1, ..., 2^{d-1}\}$ **do**
    i) For each feature $j = 1, 2, \ldots, P$, and each threshold level α, define a split as $s = (j, \alpha)$, which divides $C_l(d-1)$ into C_{left} and C_{right}:

$$C_{left}(s) = \{z_j \leq \alpha\} \cap C_l(d-1); \quad C_{right}(s) = \{z_j > \alpha\} \cap C_l(d-1),$$

    where z_j denotes the jth covariate.
    ii) Define the impurity function:

$$\begin{aligned}\mathcal{L}(C, C_{left}, C_{right}) =& \frac{|C_{left}|}{|C|} H(C_{left}) + \frac{|C_{right}|}{|C|} H(C_{right}), \text{ where} \\ H(C) =& \frac{1}{|C|} \sum_{z_{i,t} \in C} (r_{i,t+1} - \theta)^2, \quad \theta = \frac{1}{|C|} \sum_{z_{i,t} \in C} r_{i,t+1},\end{aligned}$$

    and $|C|$ denotes the number of observations in set C.
    iii) Select the optimal split:

$$s^* \leftarrow \underset{s}{\text{argmin}}\ \mathcal{L}(C(s), C_{left}(s), C_{right}(s)).$$

    iv) Update the nodes:

$$C_{2l-1}(d) \leftarrow C_{left}(s^*), \quad C_{2l}(d) \leftarrow C_{right}(s^*).$$

  end
end
Result: The output of a regression tree is given by:

$$g(z_{i,t}; \theta, L) = \sum_{k=1}^{2^L} \theta_k \mathbf{1}\left\{z_{i,t} \in C_k(L)\right\}, \text{ where } \theta_k = \frac{1}{|C_k(L)|} \sum_{z_{i,t} \in C_k(L)} r_{i,t+1}.$$

For a single binary complete regression tree $\mathcal{T}$ of depth L, the VIP for the covariate z_j is

$$\text{VIP}(z_j, \mathcal{T}) = \sum_{d=1}^{L-1} \sum_{i=1}^{2^{d-1}} \Delta im\left(C_i(d-1), C_{2i-1}(d), C_{2i}(d)\right) \mathbf{1}\{z_j \in \mathcal{T}(i,d)\},$$

where $\mathcal{T}(i,d)$ represents the covariate on the i-th (internal) node of depth d, which splits $C_i(d-1)$ into two sub-regions $\{C_{2i-1}(d), C_{2i}(d)\}$, and $\Delta im(\cdot,\cdot,\cdot)$ is defined by:

$$\Delta im\left(C, C_{left}, C_{right}\right) = H(C) - \mathcal{L}(C, C_{left}, C_{right}).$$

Algorithm 3: Gradient Boosted Tree

Initialize the predictor as $\widehat{g}_0(\cdot) = 0$;

for b *from* 1 *to* B **do**

Compute for each $i = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T$, the negative gradient of the loss function $l(\cdot, \cdot)$:[a]

$$\varepsilon_{i,t+1} \leftarrow -\frac{\partial l(r_{i,t+1}, g)}{\partial g} \Big|_{g=\widehat{g}_{b-1}(z_{i,t})}.$$

Grow a (shallow) regression tree of depth L with dataset $\{(z_{i,t}, \varepsilon_{i,t+1}) : \ \forall i, \ \forall t\}$

$$\widehat{f}_b(\cdot) \leftarrow g(z_{i,t}; \theta, L).$$

Update the model by

$$\widehat{g}_b(\cdot) \leftarrow \widehat{g}_{b-1}(\cdot) + \nu \widehat{f}_b(\cdot),$$

where $\nu \in (0, 1]$ is a tuning parameter that controls the step length.

end

Result: The final model output is

$$\widehat{g}_B(z_{i,t}; B, \nu, L) = \sum_{b=1}^{B} \nu \widehat{f}_b(\cdot).$$

[a] The typical choice of $l(\cdot, \cdot)$ for regression is l_2 or Huber loss, whereas for classification, it is more common to use the following loss function:

$$l(d, g(\cdot)) = \log_2 \left(1 + \exp\left(-2(2d - 1)g(\cdot)\right)\right).$$

Algorithm 4: Random Forest

for b *from* 1 *to* B **do**

Generate Bootstrap samples $\{(z_{i,t}, r_{i,t+1}), (i, t) \in Bootstrap(b)\}$ from the original dataset, for which a tree is grown using Algorithm 2. At each step of splitting, use only a random subsample, say $\sqrt{P}$, of all features. Write the resulting bth tree as:

$$\widehat{g}_b(z_{i,t}; \widehat{\theta}_b, L) = \sum_{k=1}^{2^L} \theta_b^{(k)} \mathbf{1}\left\{z_{i,t} \in C_k(L)\right\}.$$

end

Result: The final random forest output is given by the average of the outputs of all B trees.

$$\widehat{g}(z_{i,t}; L, B) = \frac{1}{B} \sum_{b=1}^{B} \widehat{g}_b(z_{i,t}; \widehat{\theta}_b, L).$$

A.3 Neural Networks

It is common to fit the neural network using stochastic gradient descent (SGD), see, e.g., Goodfellow et al. (2016). We adopt the adaptive moment estimation algorithm (Adam), an efficient version of the SGD introduced by Kingma and Ba (2014). Adam computes adaptive learning rates for individual parameters using estimates of first and second moments of the gradients. We denote the loss function as $\mathcal{L}(\theta;\cdot)$ and write $\mathcal{L}(\theta;\cdot) = \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_t(\theta;\cdot)$, where $\mathcal{L}_t(\theta;\cdot)$ is the penalized cross-sectional average prediction error at time t. At each step of training, a batch sent to the algorithm is the entire cross-section of observations at a time t, for $t = 1, 2, \ldots, T$. We reverse the order of time from T to 1 to prioritize more recent information. Algorithm 6 is the early stopping algorithm that can be used in combination with many optimization routines, including Adam. Algorithm 7 gives the Batch-Normalization transform, which we apply to each activation after ReLU transformation. Any neuron that previously receives a batch of x as the input now receives $\mathrm{BN}_{\gamma,\beta}(x)$ instead, where γ and β are additional parameters to be optimized.

Algorithm 5: Adam for Stochastic Gradient Descent (SGD)

Initialize the parameter vector θ_0. Set $m_0 = 0, v_0 = 0, t = 0$.

while θ_t *not converged* **do**

$t \leftarrow t + 1$.

$g_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta;\cdot)\big|_{\theta=\theta_{t-1}}$.

$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$.

$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t \odot g_t$.[a]

$\hat{m}_t \leftarrow m_t / (1-(\beta_1)^t)$.

$\hat{v}_t \leftarrow v_t / (1-(\beta_2)^t)$.

$\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t \oslash (\sqrt{\hat{v}_t} + \epsilon)$.

end

Result: The final parameter estimate is θ_t.

[a] $\odot$ and $\oslash$ denote element-wise multiplication and division, respectively.

Algorithm 6: Early Stopping

Initialize $j = 0$, $\epsilon = \infty$ and select the patience parameter p.

while $j < p$ **do**

Update θ using the training algorithm (e.g., the steps inside the while loop of Algorithm 5 for h steps).

Calculate the prediction error from the validation sample, denoted as ϵ'.

if $\epsilon' < \epsilon$ **then**

$j \leftarrow 0$.

$\epsilon \leftarrow \epsilon'$.

$\theta' \leftarrow \theta$.

else

$j \leftarrow j + 1$.

end

end

Result: The final parameter estimate is θ'.

Algorithm 7: Batch Normalization (for one Activation over one Batch)

Input: Values of x for each activation over a batch $\mathcal{B} = \{x_1, x_2, \ldots, x_N\}$.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_{\mathcal{B}})^2$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta := \mathrm{BN}_{\gamma,\beta}(x_i)$$

Result: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i) : i = 1, 2 \ldots, N\}$.

Table A.1: Details of the Characteristics

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

Note: This table lists the characteristics we use in the empirical study. The data are collected in Green et al. (2013).

Table A.1: Details of the Characteristics (Continued)

No.	Acronym	Firm characteristic	Paper's author(s)	Year, Journal	Data Source	Frequency
32	egr	Growth in common shareholder equity	Richardson, Sloan, Soliman & Tuna	2005, JAE	Compustat	Annual
33	ep	Earnings to price	Basu	1977, JF	Compustat	Annual
34	gma	Gross profitability	Novy-Marx	2013, JFE	Compustat	Annual
35	grCAPX	Growth in capital expenditures	Anderson & Garcia-Feijoo	2006, JF	Compustat	Annual
36	grltnoa	Growth in long term net operating assets	Fairfield, Whisenant & Yohn	2003, TAR	Compustat	Annual
37	herf	Industry sales concentration	Hou & Robinson	2006, JF	Compustat	Annual
38	hire	Employee growth rate	Bazdresch, Belo & Lin	2014, JPE	Compustat	Annual
39	idiovol	Idiosyncratic return volatility	Ali, Hwang & Trombley	2003, JFE	CRSP	Monthly
40	ill	Illiquidity	Amihud	2002, JFM	CRSP	Monthly
41	indmom	Industry momentum	Moskowitz & Grinblatt	1999, JF	CRSP	Monthly
42	invest	Capital expenditures and inventory	Chen & Zhang	2010, JF	Compustat	Annual
43	IPO	New equity issue	Loughran, Ritter & Ritter	1995, JF	CRSP	Monthly
44	lev	Leverage	Bhandari	1988, JF	Compustat	Annual
45	lgr	Growth in long-term debt	Richardson, Sloan, Soliman & Tuna	2005, JAE	Compustat	Annual
46	maxret	Maximum daily return	Bali, Cakici & Whitelaw	2011, JFE	CRSP	Monthly
47	mom12m	12-month momentum	Jegadeesh	1990, JF	CRSP	Monthly
48	mom1m	1-month momentum	Jegadeesh & Titman	1993, JF	CRSP	Monthly
49	mom36m	36-month momentum	Jegadeesh & Titman	1993, JF	CRSP	Monthly
50	mom6m	6-month momentum	Jegadeesh & Titman	1993, JF	CRSP	Monthly
51	ms	Financial statement score	Mohanram	2005, RAS	Compustat	Quarterly
52	mvel1	Size	Banz	1981, JFE	CRSP	Monthly
53	mve_ia	Industry-adjusted size	Asness, Porter & Stevens	2000, WP	Compustat	Annual
54	nincr	Number of earnings increases	Barth, Elliott & Finn	1999, JAR	Compustat	Quarterly
55	operprof	Operating profitability	Fama & French	2015, JFE	Compustat	Annual
56	orgcap	Organizational capital	Eisfeldt & Papanikolaou	2013, JF	Compustat	Annual
57	pchcapx_ia	Industry adjusted % change in capital expenditures	Abarbanell & Bushee	1998, TAR	Compustat	Annual
58	pchcurrat	% change in current ratio	Ou & Penman	1989, JAE	Compustat	Annual
59	pchdepr	% change in depreciation	Holthausen & Larcker	1992, JAE	Compustat	Annual
60	pchgm_pchsale	% change in gross margin - % change in sales	Abarbanell & Bushee	1998, TAR	Compustat	Annual
61	pchquick	% change in quick ratio	Ou & Penman	1989, JAE	Compustat	Annual
62	pchsale_pchinvt	% change in sales - % change in inventory	Abarbanell & Bushee	1998, TAR	Compustat	Annual
63	pchsale_pchrect	% change in sales - % change in A/R	Abarbanell & Bushee	1998, TAR	Compustat	Annual

Table A.1: Details of the Characteristics (Continued)

No.	Acronym	Firm characteristic	Paper's author(s)	Year, Journal	Data Source	Frequency
64	pchsale_pchxsga	% change in sales - % change in SG&A	Abarbanell & Bushee	1998, TAR	Compustat	Annual
65	pchsaleinv	% change sales-to-inventory	Ou & Penman	1989, JAE	Compustat	Annual
66	pctacc	Percent accruals	Hafzalla, Lundholm & Van Winkle	2011, TAR	Compustat	Annual
67	pricedelay	Price delay	Hou & Moskowitz	2005, RFS	CRSP	Monthly
68	ps	Financial statements score	Piotroski	2000, JAR	Compustat	Annual
69	quick	Quick ratio	Ou & Penman	1989, JAE	Compustat	Annual
70	rd	R&D increase	Eberhart, Maxwell & Siddique	2004, JF	Compustat	Annual
71	rd_mve	R&D to market capitalization	Guo, Lev & Shi	2006, JBFA	Compustat	Annual
72	rd_sale	R&D to sales	Guo, Lev & Shi	2006, JBFA	Compustat	Annual
73	realestate	Real estate holdings	Tuzel	2010, RFS	Compustat	Annual
74	retvol	Return volatility	Ang, Hodrick, Xing & Zhang	2006, JF	CRSP	Monthly
75	roaq	Return on assets	Balakrishnan, Bartov & Faurel	2010, JAE	Compustat	Quarterly
76	roavol	Earnings volatility	Francis, LaFond, Olsson & Schipper	2004, TAR	Compustat	Quarterly
77	roeq	Return on equity	Hou, Xue & Zhang	2015, RFS	Compustat	Quarterly
78	roic	Return on invested capital	Brown & Rowe	2007, WP	Compustat	Annual
79	rsup	Revenue surprise	Kama	2009, JBFA	Compustat	Quarterly
80	salecash	Sales to cash	Ou & Penman	1989, JAE	Compustat	Annual
81	saleinv	Sales to inventory	Ou & Penman	1989, JAE	Compustat	Annual
82	salerec	Sales to receivables	Ou & Penman	1989, JAE	Compustat	Annual
83	secured	Secured debt	Valta	2016, JFQA	Compustat	Annual
84	securedind	Secured debt indicator	Valta	2016, JFQA	Compustat	Annual
85	sgr	Sales growth	Lakonishok, Shleifer & Vishny	1994, JF	Compustat	Annual
86	sin	Sin stocks	Hong & Kacperczyk	2009, JFE	Compustat	Annual
87	SP	Sales to price	Barbee, Mukherji, & Raines	1996, FAJ	Compustat	Annual
88	std_dolvol	Volatility of liquidity (dollar trading volume)	Chordia, Subrahmanyam & Anshuman	2001, JFE	CRSP	Monthly
89	std_turn	Volatility of liquidity (share turnover)	Chordia, Subrahmanyam, &Anshuman	2001, JFE	CRSP	Monthly
90	stdacc	Accrual volatility	Bandyopadhyay, Huang & Wirjanto	2010, WP	Compustat	Quarterly
91	stdcf	Cash flow volatility	Huang	2009, JEF	Compustat	Quarterly
92	tang	Debt capacity/firm tangibility	Almeida & Campello	2007, RFS	Compustat	Annual
93	tb	Tax income to book income	Lev & Nissim	2004, TAR	Compustat	Annual
94	turn	Share turnover	Datar, Naik & Radcliffe	1998, JFM	CRSP	Monthly
95	zerotrade	Zero trading days	Liu	2006, JFE	CRSP	Monthly