Empirical Asset Pricing via Machine Learning*

Shihao Gu
University of Chicago

Bryan Kelly
Yale University, NBER, and
AQR Capital Management

Dacheng Xiu
University of Chicago

This Version: April 9, 2018

Abstract

We synthesize the field of machine learning with the canonical problem of empirical asset pricing: Measuring asset risk premia. In the familiar empirical setting of cross section and time series stock return prediction, we perform a comparative analysis of methods in the machine learning repertoire, including generalize linear models, dimension reduction, boosted regression trees, random forests, and neural networks. At the broadest level, we find that machine learning offers an improved description of asset price behavior relative to traditional methods. 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 nonlinear 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 includes variations on momentum, liquidity, and volatility. 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: Machine Learning, Return Prediction, Cross-Section of Returns, Ridge Regression, (Group) Lasso, Elastic Net, Random Forest, Gradient Boosting, (Deep) Neural Networks, Fintech

*We benefitted from discussions with Joseph Babcock, Rob Engle, Lasse Pedersen, Guofu Zhou, and seminar and conference participants at Erasmus School of Economics, National University of Singapore, Fannie Mae, Tsinghua Workshop on Big Data and Internet Economics, and the 2017 Conference on Financial Predictability and Data Science. We gratefully acknowledge the computing support from the Research Computing Center at the University of Chicago.

1 Introduction

In this article, we conduct a comparative analysis of 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. This accuracy 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 of 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 efficiency of markets leads return variation to be dominated by unforecastable news that obscures risk premia. 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. Relative to traditional empirical methods in asset pricing, machine learning accommodates a far more expansive list of potential predictor variables and richer specifications of functional form. It is this flexibility that allows us to push the frontier of risk premium measurement. 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 researchers to understand the usefulness of machine learning in the field of asset pricing is to apply and compare the performance of each of its methods in familiar empirical problems.

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 a vast number of potential 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 our 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 machine learning tools designed to approximate an optimal specification with manageable computational cost.

1.3 Why Apply Machine Learning to Asset Pricing?

A number of aspects of empirical asset pricing make it a particularly attractive field 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. Measurement of an asset's risk premium is fundamentally a problem of prediction—the risk premium is the conditional expectation of a future realized excess return. 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 candidate conditioning variables for the risk premium is large. The profession has accumulated a staggering list of predictors that various researchers have argued 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.[1] 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. With an emphasis on variable selection and dimension reduction techniques, machine learning is well suited for such challenging prediction problems by reducing degrees of freedom and condensing redundant variation among predictors.

3) Further complicating the problem is ambiguity regarding functional forms through which the high-dimensional predictor set enter into risk premia. Should they enter linearly? If nonlinearities 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 of conditioning variables and functional forms. Three 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, with methods ranging from generalized linear models to regression trees and neural networks, machine learning is explicitly designed to approximate complex nonlinear 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 and false discovery.

[1] 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.

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.[2] This includes linear regression, generalized linear models with penalization, dimension reduction via principal components regression (PCR) and partial least squares (PLS), regression trees (including boosted trees and random forests), and neural networks. This is not an exhaustive analysis of all methods. For example, it excludes methods like support vector machines that are more commonly employed in classification problems as opposed to continuous variable prediction. 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 94 characteristics for each stock, interactions of each characteristic with eight aggregate time series variables, and 74 industry sector dummy variables, totaling more than 900 baseline signals. Some of our methods expand this predictor set much further by including nonlinear 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 digests our predictor data set, which is massive from the perspective of the existing literature, into a return forecasting model that dominates traditional approaches. The immediate implication is that machine learning aids 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 return predictors). Lewellen (2015) demonstrates 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 (more observations in terms of both dates and stocks) than that studied in Lewellen (2015), the out-of-sample R^2 from the benchmark model is 0.16% per month for the panel of individual stock returns. When we expand the OLS panel model to include our set of 900+ predictors, predictability vanishes immediately—the R^2 drops deeply into negative territory. This is not surprising. With so many parameters to estimate, efficiency of OLS regression deteriorates precipitously and therefore produces forecasts that are highly unstable out-of-sample.

Vast predictor sets are viable for linear prediction when either penalization or dimension reduction

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

is used. Our first evidence that the machine learning toolkit aids in return prediction emerges from the fact that straightforward "elastic net," which uses parameter shrinkage and variable selection to limit the regression's degrees of freedom, solves the OLS inefficiency problem. In the 900+ predictor regression, elastic net pulls the out-of-sample R^2 into positive territory at 0.09% per month. Principal component regression (PCR) and partial least square (PLS), which reduces the dimension of the predictor set to a few linear combinations of predictors, further raise the out-of-sample R^2 to 0.18% and 0.28%, respectively. This is in spite of the presence of many likely "fluke" predictors that contribute pure noise to the large model. 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 over-parameterization can be controlled.

Allowing for nonlinearities substantially improves predictions. Next, we expand the model to accommodate nonlinear predictive relationships via generalize linear models, regression trees, and neural networks. We find that trees and neural nets unambiguously improve return prediction with monthly stock-level R^2's between 0.27% and 0.39%. But the generalized linear model, which introduces nonlinearity via spline functions of each individual baseline predictor (but with no predictor interactions), fails to robustly outperform the linear specification. This suggests that allowing for (potentially complex) interactions among the baseline predictors is a crucial aspect of nonlinearities in the expected return function. As part of our analysis, we discuss why generalized linear models are comparatively poorly suited for capturing predictor interactions.

Shallow learning outperforms deeper learning. When we consider a range of neural networks from very shallow (a single hidden layer) to deeper networks (up to five hidden layers), we find that neural network performance peaks at three hidden layers then declines as more layers are added. Likewise, the 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 nonlinear methods and the benchmark widens when predicting portfolio returns. We build bottom-up portfolio-level return forecasts from the stock-level forecasts produced by our models. Consider, for example, bottom-up forecasts of the S&P 500 portfolio return. By aggregating stock-level forecasts from the benchmark three-characteristic OLS model, we find a monthly S&P 500 predictive R^2 of -0.11%. The bottom-up S&P 500 forecast from the generalized linear model, in contrast, delivers an R^2 of 0.86%. Trees and neural networks improve upon this further, generating monthly out-of-sample R^2's between 1.39% to 1.80% per month. The same pattern emerges for forecasting a variety of characteristic factor portfolios, such as those formed on the basis of size, value, investment, profitability, and momentum. In particular, neural networks produce positive out-of-sample predictive R^2's for all of the factor portfolios we consider.

More pronounced predictive power at the portfolio level versus the stock level is driven by the fact that individual stock returns behave erratically for some of the smallest and least liquid stocks in our sample. Aggregating into portfolios averages out much of the unpredictable stock-level noise

to better detect the predictive benefits of machine learning.

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

Learning through simulation. In Appendix A we perform Monte Carlo simulations that support the above interpretations of our analysis. We apply machine learning to simulated data from two different data generating processes. Both generate data from a high dimensional predictor set. But in one data generating process, individual predictors enter only linearly and additively, while in the other predictors can enter through nonlinear transformations and via pairwise interactions. When we apply our machine learning repertoire to the simulated datasets, we find that linear and generalized linear methods dominate in the linear and uninteracted setting, yet tree-based methods and neural networks significantly outperform in the nonlinear and interactive setting.

1.6 What Machine Learning Cannot Do

Machine learning has great potential for improving risk premium *measurements*, which is fundamentally a problem of prediction. It amounts to best approximating the conditional expectation $\mathrm{E}(r_{i,t+1}|\mathcal{F}_t)$, where $r_{i,t+1}$ is an asset's return in excess of the risk-free rate, and $\mathcal{F}_t$ is the true and unobservable information set of market participants. This is a domain in which machine learning algorithms excel.

But, ultimately, these improved predictions are only measurements. The measurements do not tell us about economic *mechanisms* or *equilibria*. Machine learning methods on their own cannot identify deep fundamental associations among asset prices and conditioning variables. When the objective is to understand economic mechanisms, machine learning may still be useful. It requires the economist to add structure—to build a hypothesized mechanism into the estimation problem—and decide how to introduce a machine learning algorithm subject to this structure. A nascent literature has begun to make progress marrying machine learning with equilibrium asset pricing (for example, Kelly and Pruitt, 2013; Feng et al., 2017; Kelly et al., 2017), and this remains an exciting direction for future research.

1.7 Literature

Our work extends the empirical literature on stock return prediction, which comes in two basic strands. The first strand models differences in expected returns across stocks as a function of stock-level characteristics, and is exemplified by Fama and French (2008) and Lewellen (2015). The typical approach in this literature runs cross-sectional regressions 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 strand forecasts the time series of returns and is surveyed

by Koijen and Nieuwerburgh (2011) and Rapack and Zhou (2013). This literature typically conducts time series regressions of broad aggregate portfolio returns on a small number of 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 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 and false discovery problem). Our primary contribution is to demonstrate potent return predictability that is 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; 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. Sirignano et al. (2016) estimate a deep neural network for mortgage prepayment, delinquency, and foreclosure. 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 and selection methods to, respectively, approximate a stochastic discount factor and a nonlinear 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 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 nonlinear 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 our specific implementation choices in Appendix C 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 $t = 1, ..., T$. For ease of presentation, we assume a balanced panel of stocks, and defer the discussion on missing data to Section 3.1. 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 i^{th}.

2.1 Sample Splitting and Tuning via Validation

Before discussing specific models and regularization approaches, it is important to understand the importance of designing disjoint sub-samples for estimation and testing, and to introduce the notion of "hyperparameter tuning."

The regularization procedures discussed below, which are machine learning's primary defense against overfitting, rely on a choice of hyperparameters (or, synonymously, "tuning parameters"). These are critical to the performance of machine learning methods as they control the model complexity. Hyperparameters 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 "tune" hyperparameters so that they are optimized for out-of-sample performance.

We follow the most common approach in the literature and select tuning parameters adaptively from the data in a validation sample. In particular, 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 tuning the hyperparameters. 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 hyperparameters that optimize the validation objective (at each step re-estimating the model from the training data subject to the prevailing hyperparameter values).

Tuning parameters are chosen from the validation sample taking into account estimated parameters, but the parameters are estimated from the training data alone. The idea of validation is to simulate an out-of-sample test of the model. Hyperparameter tuning amounts to searching 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. Further details of our sample splitting scheme are provided in Appendix B.

2.2 Simple Linear

We begin our model description 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, we use it as 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,[3]

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

This model imposes a simple regression specification and does not allow for nonlinear effects or interactions between 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 offers analytical estimates and thus avoids sophisticated optimization and computation.

[3]More precisely, the OLS model can be written as, $r_{i,t+1} = z_{i,t}'\theta + e_{i,t+1}$, where $e_{i,t+1} = \epsilon_{i,t+1} + g^\star(z_{i,t}) - z_{i,t}\theta$. This is the standard best linear predictor framework, which does not require $g^\star(z_{i,t})$ to be linear. θ is identified via $\mathrm{E}(e_{i,t+1}z_{i,t}) = 0$.

2.2.1 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 $w_{i,t}$ inversely proportional to the number of stocks at time t. This imposes that every month has the same contribution to the model regardless of how many stocks are available that month. This also amounts to equal-weighting the squared loss of all stocks available at time t. Another 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.[4]

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.[5] 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.[6]

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

[4] 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.

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

[6] 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 hyperparameter.

we study the predictive benefits of robust loss functions, as well as alternative weighting schemes, for multiple machine learning methods.

2.3 Penalized Linear

The simple linear model is bound to fail in the presence of many predictors. When the number of predictors P approaches the number of observations T, the linear model becomes inefficient or even inconsistent.[7] It begins to overfit noise rather than extracting signal. This is particularly troublesome for the problem of return prediction where the signal-to-noise ratio is notoriously low.

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 our penalized linear model is the same as the simple linear model in equation (3). That is, it continues to consider only the baseline, untransformed predictors. 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 involves two non-negative hyperparameters, λ and ρ, and includes two well known regularizers as special cases. The $\rho = 0$ case 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 $\rho = 1$ case 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 selection.

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 C.1 for more detail).

[7] We deliberately compare P with T, instead of with NT, because stock returns share strong cross-sectional dependence, limiting the incremental information contained in new cross section observations.

2.4 Dimension Reduction: PCR and PLS

Penalized linear models use shrinkage and variable selection to manage high dimensionality by forcing the coefficients on most regressors near or exactly to zero. This can produce suboptimal forecasts when predictors are highly correlated. A transparent illustration of this problem would be a case in which all of the predictors are equal to the forecast target plus an iid noise term. In this situation, choosing a subset of predictors via LASSO penalty is inferior to taking a simple average of the predictors and using this as the sole predictor in a univariate regression.

The idea of predictor averaging, as opposed to predictor selection, is the essence of dimension reduction. Averaging helps reduce noise to better isolate the signal in predictors, and discerningly selected linear combinations help de-correlate otherwise highly dependent predictors. Two classic dimension reduction techniques are principal components regression (PCR) and partial least squares (PLS).

PCR consists of a two-step procedure. In the first step, principal components analysis (PCA) combines regressors into a small set of linear combinations that best preserve the covariance structure among the predictors. In the second step, a leading few components are used in standard predictive regression. That is, PCR regularizes the prediction problem by zeroing out coefficients on higher order components. Contrast this with ridge regression, which essentially shrinks the coefficients on all components towards zero.

A drawback of PCR is that it fails to incorporate the ultimate statistical objective—forecasting returns—in the dimensionality reduction step. PCA condenses data into components based on the covariation *among* the predictors. This happens prior to the forecasting step and without consideration of how predictors associate with future returns.

In contrast, partial least squares performs dimension reduction by directly exploiting covariation of predictors with the forecast target.[8] PLS regression proceeds as follows. For each predictor j, estimate its univariate return prediction coefficient via OLS. This coefficient, denoted φ_j, reflects the "partial" sensitivity of returns to each predictor j. Next, average all predictors into a single aggregate component with weights proportional to φ_j, placing the highest weight on the strongest univariate predictors, and the least weight on the weakest. In this way, PLS performs its dimension reduction with the ultimate forecasting objective in mind. To form more than one predictive component, the target and all predictors are orthogonalized with respect to previously constructed components, and the above procedure is repeated on the orthogonalized dataset. This is iterated until the desired number of PLS components is reached.

Model. Our implementation of PCR and PLS begins from the vectorized version of the linear model in equations (1)–(3). In particular, we reorganize the linear regression $r_{i,t+1} = z_{i,t}'\theta + \epsilon_{i,t+1}$ as

$$R = Z\theta + E, \tag{9}$$

where R is the $NT \times 1$ vector of $r_{i,t+1}$, Z is the $NT \times P$ matrix of stacked predictors $z_{i,t}$, and E is

[8] See Kelly and Pruitt (2013, 2015) for asymptotic theory of PLS regression and its application to forecasting risk premia in financial markets.

a $NT \times 1$ vector of residuals $\epsilon_{i,t+1}$.

PCR and PLS take the same general approach to reducing the dimensionality. They both condense the set of predictors from dimension P to a much smaller number K of linear combinations of predictors. Thus, the forecasting model for both methods is written as

$$R = (Z\Omega_K)\theta + E. \tag{10}$$

Ω_K is $P \times K$ matrix with columns $w_1, w_2, \ldots, w_K$. Each w_j is the set of linear combination weights used to create the j^{th} predictive components, thus $Z\Omega_K$ is the dimension-reduced version of the original predictor set. Likewise, the predictive coefficient θ is now a $K \times 1$ vector rather than $P \times 1$.

Objective Function and Computational Algorithm. PCR chooses the combination weights Ω_K recursively. The j^{th} linear combination solves

$$w_j = \arg\max_w \text{Var}(Zw), \quad \text{s.t.} \quad w'w = 1, \quad w'Z'Z[w_1, .., w_{j-1}] = 0. \tag{11}$$

Intuitively, PCR seeks the K linear combinations of Z that most faithfully mimic the full predictor set. The objective illustrates that the choice of components is not based on the forecasting objective at all. Instead, the emphasis of PCR is on finding components that retain the most possible common variation within the predictor set. The well known solution for (11) computes Ω_K via singular value decomposition of Z, and therefore the PCR algorithm is extremely efficient from a computational standpoint.

In contrast to PCR, the PLS objective seeks K linear combinations of Z that have maximal predictive association with the forecast target. The weights used to construct j^{th} PLS component solve

$$w_j = \arg\max_w \ \text{Corr}^2(R, Zw)\text{Var}(Zw), \quad \text{s.t.} \quad w'w = 1, \quad w'Z'Z[w_1, .., w_{j-1}] = 0. \tag{12}$$

This objective highlights the main distinction between PCR and PLS. PLS is willing to sacrifice how accurately $Z\Omega_K$ approximates Z in order to find components with more potent return predictability. The problem in (12) can be efficiently solved using a number of similar routines, perhaps the most prominent being the SIMPLS algorithm of de Jong (1993).

Finally, given a solution for Ω_K, θ is estimated in both PCR and PLS via OLS regression of R on $Z\Omega_K$. And for both models, K is a hyperparameter that can be determined adaptively from the validation sample.

2.5 Generalized Linear

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 nonlinear, restricting the functional form to be linear introduces approximation error due to model misspecification. Let $g^\star(z_{i,t})$ denote the true model and $g(z_{i,t}; \theta)$ the functional form specified by the econometrician. And let $g(z_{i,t}; \widehat{\theta})$ and $\widehat{r}_{i,t+1}$ denote 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. Approximation error is directly controlled by the econometrician, and is potentially reducible by incorporating more flexible specifications that improve the model's ability to approximate the true model. But additional flexibility raises the risk of overfitting and destabilizing the model out-of-sample. In this and the following subsections, we introduce non-parametric models of $g(\cdot)$ with increasing degrees of flexibility, each complemented by regularization methods to mitigate overfit.

Model. The first and most straightforward non-parametric approach that we consider is the generalized linear model. It introduces nonlinear transformations of the original predictors as new additive terms in the model. This essentially builds an expanded set of predictors in an otherwise linear model. Generalized linear models are thus the closest nonlinear counterparts to the linear approaches in Sections 2.2 and 2.3.

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

$$g(z;\theta,p(\cdot)) = \sum_{j=1}^{P} p(z_j)'\theta_j, \tag{13}$$

where $p(\cdot) = (p_1(\cdot), p_2(\cdot), \ldots, p_K(\cdot))'$ is a vector of basis functions, and the parameters are now a $K \times N$ matrix $\theta = (\theta_1, \theta_2, \ldots, \theta_N)$. There are many potential choices for spline functions. We adopt a spline series of order two: $\big(1, z, (z-c_1)^2, (z-c_2)^2, \ldots, (z-c_{K-2})^2\big)$, where c_1, c_2, $\ldots c_{K-2}$ are knots.

Objective Function and Computational Algorithm. Because higher order terms enter additively, forecasting with the generalized linear model can be approached with the same estimation tools as in Section 2.2. 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

$$\phi(\theta;\lambda,K) = \lambda \sum_{j=1}^{P} \left(\sum_{k=1}^{K} \theta_{j,k}^2 \right)^{1/2} . \tag{14}$$

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. In this case there are two tuning parameters, λ and K.[9]

2.6 Boosted Regression Trees and Random Forests

The model in (13) captures individual predictors' nonlinear 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 linear model becomes computationally infeasible.[10]

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" in a sequence of steps. At each step, a new "branch" sorts the data leftover from the preceding step into 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.

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

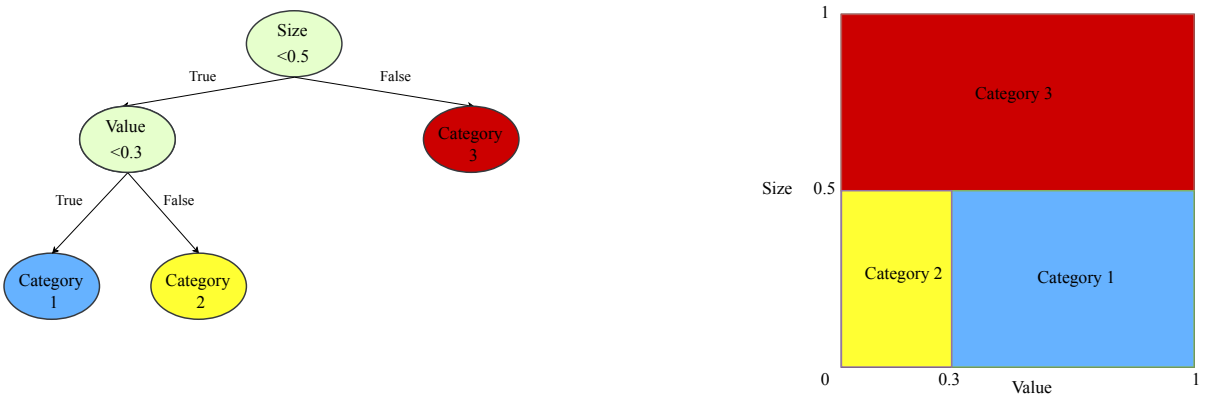
$$g(z_{i,t};\theta,K,L)=\sum_{k=1}^{K}\theta_k\mathbf{1}_{\{z_{i,t}\in C_k(L)\}}, \tag{15}$$

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

[9] For additional details, see Appendix C.1.

[10] 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.

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.

average of outcomes within the partition.[11] 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 best discriminate among the potential outcomes. The specific predictor variable upon which a branch is based, and the specific value where the branch is split, is chosen to minimize forecast error. The expanse of potential tree structures, however, 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 C.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.[12] 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:

$$H(\theta, C) = \frac{1}{|C|}\sum_{z_{i,t}\in C}(r_{i,t+1}-\theta)^2, \tag{16}$$

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

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

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

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 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.[13]

Boosting. The first regularization method is "boosting," which recursively combines forecasts from many over-simplified trees.[14] 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 our boosting procedure, typically referred to as gradient boosted regression trees (GBRT), are described in Algorithm 3 of Appendix C.2. It starts by fitting a shallow tree (e.g., with depth $L = 1$). This over-simplified 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 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 from 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, a 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 using a method known as "dropout," which considers only a randomly drawn subset of predictors for splitting at each potential

[13] 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.

[14] 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.

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.

2.7 Neural Networks

The final nonlinear 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 nonlinear 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 nonlinearly 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.

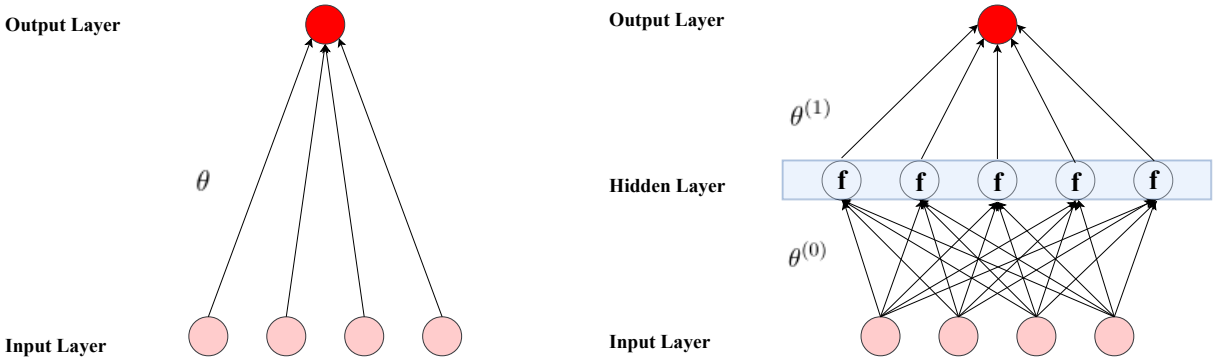
The number of units in the input layer is equal to the dimension of the predictors, which we set to four in this example (denoted $z_1, ..., z_4$). The left panel shows the simplest possible network that has no hidden layers. Each of the predictor signals is amplified or attenuated according to a five-dimensional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

The model incorporates more flexible predictive associations by adding hidden layers between the inputs and output. The right panel of Figure 2 shows an example with one hidden layer that contains five neurons. Each neuron draws information linearly from all of the input 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 in the hidden layer transforms inputs into an output 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 ultimate 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 $31 = (4+1)\times 5 + 6$ parameters (five parameters to reach each neuron and six weights to aggregate the neurons into a single output).

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.

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. Despite the aforementioned "universal approximation" result that suggests the sufficiency of a single hidden layer, recent literature has shown that deeper networks have the advantage of achieving the same accuracy with substantially smaller number of parameters, hence resulting in a more parsimonious representation.[15]

In small data sets, simple networks with only a few layers and nodes often perform best, as it becomes difficult to support a rich parameterization without a very large number of observations. Selecting a successful network architecture by cross-validation is in general a difficult task. Thanks to recent advances in training and regularizing neural networks, which we discuss in detail below, we only need to determine the maximum number of neurons for each layer as well as the total number of layers.

We consider a variety of network architectures having up to five hidden layers. Our shallowest neural network has a single hidden layer of 32 neurons, which we denoted NN1. Next, NN2 has two hidden layers with 32 and 16 neurons, respectively; NN3 has three hidden layers with 32, 16, and 8 neurons, respectively; NN4 has four hidden layers with 32, 16, 8, 4 neurons, respectively; and NN5 has five hidden layers with 32, 16, 8, 4, and 2 neurons, respectively. We choose the number of neurons in each layer according to the geometric pyramid rule (see Masters, 1993). All architectures are fully connected: each unit receives an input from all units in the layer below. By comparing the performance of NN1 through NN5, we can infer the trade-offs of network depth in the return forecasting problem.

[15] Eldan and Shamir (2016) formally demonstrate that depth—even if increased by one layer—can be exponentially more valuable than increasing width in standard feed-forward neural networks. Ever since the seminal work by Hinton et al. (2006), the machine learning community has experimented and adopted deeper (and wider) networks, with as many as 152 layers for image recognition (e.g., He et al., 2016).

There are many potential choices for the nonlinear activation function (such as sigmoid, hyperbolic, softmax, and tanh). 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[16]

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

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

$$x_k^{(l)} = \text{ReLU}\left(x^{(l-1)'}\theta_k^{(l-1)}\right), \tag{17}$$

with final output

$$g(z;\theta) = x^{(L-1)'}\theta^{(L-1)}. \tag{18}$$

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.[17]

The high degree of nonlinearity and nonconvexity 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. In addition to l_1 penalization of the weight parameters, we simultaneously employ four other regularization techniques in our estimation: learning rate shrinkage, early stopping, batch normalization, and ensembles.

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

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

[17] 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.

rate (described further in Algorithm 5 of the Appendix C.3).[18]

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 close 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.[19] Early stopping can be used alone, or together with l_1-regularization as we do in this paper.

"Batch normalization" (Ioffe and Szegedy, 2015) is a simple technique for controlling the variability of predictors across different regions of the network and across different datasets. The outputs from each layer constitute a "batch." These are cross-sectionally de-meaned and variance-standardized. This means that in the right panel of Figure 2 for example, the five outputs from nodes in layer 1 are cross-sectionally de-meaned and variance standardized. This is done at all iterations of an optimization and for any dataset being considered (training, validation, or testing). A common issue limiting the effectiveness of variance standardization is so-called "covariance shift," meaning that inputs at any given level of the network can have different distributions across datasets. If these distributional differences are roughly independent of the main prediction task, then batch normalization will improves an algorithm's robustness (see Algorithm 7).

Finally, we adopt an ensemble approach in training our neural networks (see also Hansen and Salamon, 1990; Dietterich, 2000). In particular, we use multiple random seeds to initialize neural network estimation and construct predictions by averaging forecasts from all networks. Due to the stochastic nature of the optimization, this method offers large gains in reducing prediction variance. Estimation with different seeds can run independently in parallel which limits incremental computing time.

[18] 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).

[19] 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).

2.8 Performance Evaluation

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

$$R^2_{\text{oos}} = 1 - \frac{\sum_{(i,t)\in\mathcal{T}_3}(r_{i,t+1} - \widehat{r}_{i,t+1})^2}{\sum_{(i,t)\in\mathcal{T}_3} r^2_{i,t+1}}, \tag{19}$$

where $\mathcal{T}_3$ indicates that fits are only assessed on the testing subsample, whose data never enter into model estimation or tuning. The R^2_{oos} 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 or long-short portfolios, 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 three percentage points.

To make pairwise comparisons of methods, we use the Diebold and Mariano (1995) test for differences in out-of-sample predictive accuracy between two models. While time series dependence in returns is sufficiently weak, it is unlikely that the conditions of weak error dependence underlying the Diebold-Mariano test apply to our stock-level analysis due of potentially strong dependence in the cross section. We adapt Diebold-Mariano to our setting by comparing 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

$$d_{12,t+1} = \frac{1}{n_3}\sum_{i=1}^{n_3}\left(\left(\widehat{e}^{(1)}_{i,t+1}\right)^2 - \left(\widehat{e}^{(2)}_{i,t+1}\right)^2\right), \tag{20}$$

$\widehat{e}^{(1)}_{i,t+1}$ and $\widehat{e}^{(2)}_{i,t+1}$ 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. This modified Diebold-Mariano test statistic, which is now based on a single time series $d_{12,t+1}$ of error differences with little autocorrelation, is more likely to satisfy the mild regularity conditions needed for asymptotic normality, and therefore provides convenient p-values for our model comparison tests.

2.9 Variable Importance

Our goal in interpreting machine learning models is modest. We aim to identify covariates that have an important influence on the cross-section of expected returns while simultaneously controlling for the many other predictors in the system.

We discover influential covariates by ranking them according to a notion of variable importance. We denote the importance of a given input variable j as VI_j, and define it as the reduction in predictive R^2 from setting all values of predictor j to zero, while holding the remaining model estimates fixed (see Kelly et al., 2017).

3 An Empirical Study of US Equities

3.1 Data and Over-arching Model

We obtain monthly total individual equity returns from CRSP for all firms listed in the NYSE, AMEX, and NASDAQ. Our sample begins in March 1957 (the start date of the S&P 500) and ends in 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 to proxy for the risk-free rate from which we calculate individual excess returns.

In addition, we build a large collection of stock-level predictive characteristics based on the cross section of stock returns literature. These include 94 characteristics (61 of which are updated annually, 13 updated quarterly, and 20 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.4.[21]

We also construct eight macroeconomic predictors following the variable definitions detailed in Welch and Goyal (2008), including Dividend Price Ratio (dp), Earnings Price Ratio (ep), Book-to-Market Ratio (bm), Net Equity Expansion (ntis), Treasury-Bill Rate (tbl), Term Spread (tms), Default Spread (dfy), and Stock Variance (svar).[22]

All of the machine learning methods we consider are designed to approximate the over-arching empirical model $\text{E}_t(r_{i,t+1}) = g^\star(z_{i,t})$ defined in equation (2). Throughout our analysis we define the

[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 94 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.

[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.

baseline set of stock-level covariates $z_{i,t}$ as

$$z_{i,t} = x_t \otimes c_{i,t}, \tag{21}$$

where $c_{i,t}$ is a $P_c \times 1$ 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 a $P \times 1$ vector of features (with $P = P_c P_x$) for predicting individual stock returns, where these features are interactions between stock-level characteristics and macroeconomic state variables (including the baseline stock characteristics interacted with a constant). The total number of covariates is $94 \times (8 + 1) + 74 = 920$.

The over-arching model specified by (2) and (21) nests many models proposed in the literature (such as Rosenberg, 1974; 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,

$$\mathrm{E}_t(r_{i,t+1}) = \beta_{i,t}' \lambda_t. \tag{22}$$

The structure of our feature set in (21) 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 (22) becomes

$$g^\star(z_{i,t}) = \mathrm{E}_t(r_{i,t+1}) = \beta_{i,t}' \lambda_t = c_{i,t}' \theta_1' \theta_2 x_t = (x_t \otimes c_{i,t})' \operatorname{vec}(\theta_1' \theta_2) =: z_{i,t}' \theta, \tag{23}$$

where $\theta = \operatorname{vec}(\theta_1' \theta_2)$. The over-arching model is more general than this example because $g^\star(\cdot)$ is not restricted to be a linear function. Considering nonlinear $g^\star(\cdot)$ formulations, for example via generalized linear models or neural networks, essentially expands the feature set to include a variety of functional transformations of the baseline $z_{i,t}$ predictor set. Our empirical study thereby allows for general specification of the expected returns, while being agnostic about the dynamics of realized returns as well as asset pricing restrictions.

We initially 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 computationally intensive, we avoid recursively refitting models each month. Instead, we refit once every year as most of our signals are updated once per year. Each time we refit, we increase the training sample by one year. We maintain the same size of the validation sample, but rolling it forward to include the most recent twelve months.[23]

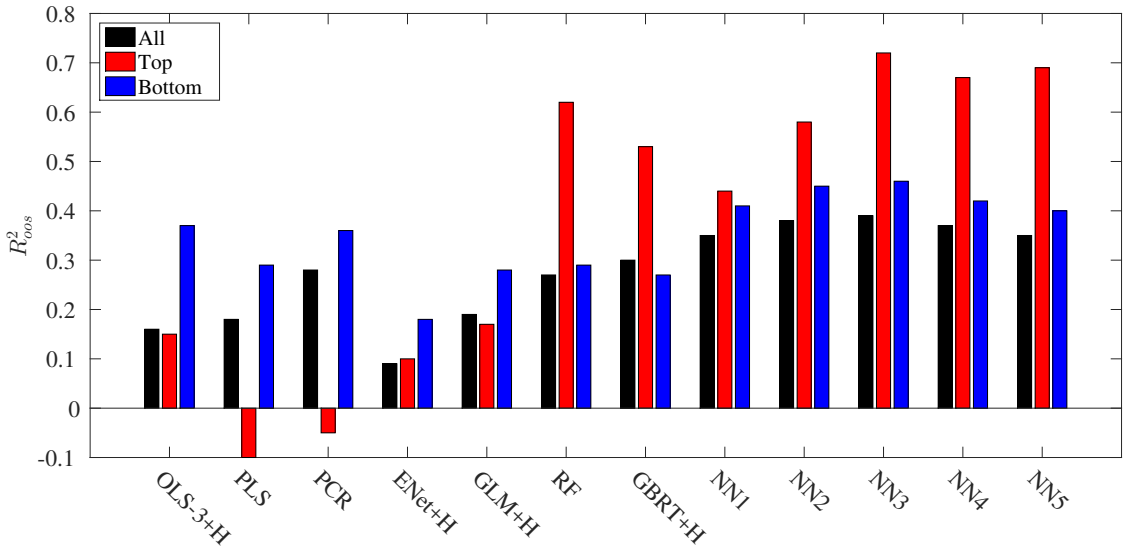
3.2 The Cross Section of Individual Stocks

Table 1 presents the comparison of machine learning techniques in terms of their out-of-sample predictive R^2. We compare thirteen models in total, including OLS with all covariates, OLS-3

[23] Note that we do not use cross-validation in order to maintain the temporal ordering of the data.

Table 1: Out-of-sample Stock-level Prediction Performance (Percentage R^2_{oos})

	OLS +H	OLS-3 +H	PLS	PCR	ENet +H	GLM +H	RF	GBRT +H	NN1	NN2	NN3	NN4	NN5
All	-4.60	0.16	0.18	0.28	0.09	0.19	0.27	0.30	0.35	0.38	0.39	0.37	0.35
Top 1000	-14.21	0.15	-0.10	-0.05	0.10	0.17	0.62	0.53	0.44	0.58	0.72	0.67	0.69
Bottom 1000	-2.13	0.37	0.29	0.36	0.18	0.28	0.29	0.27	0.41	0.45	0.46	0.42	0.40

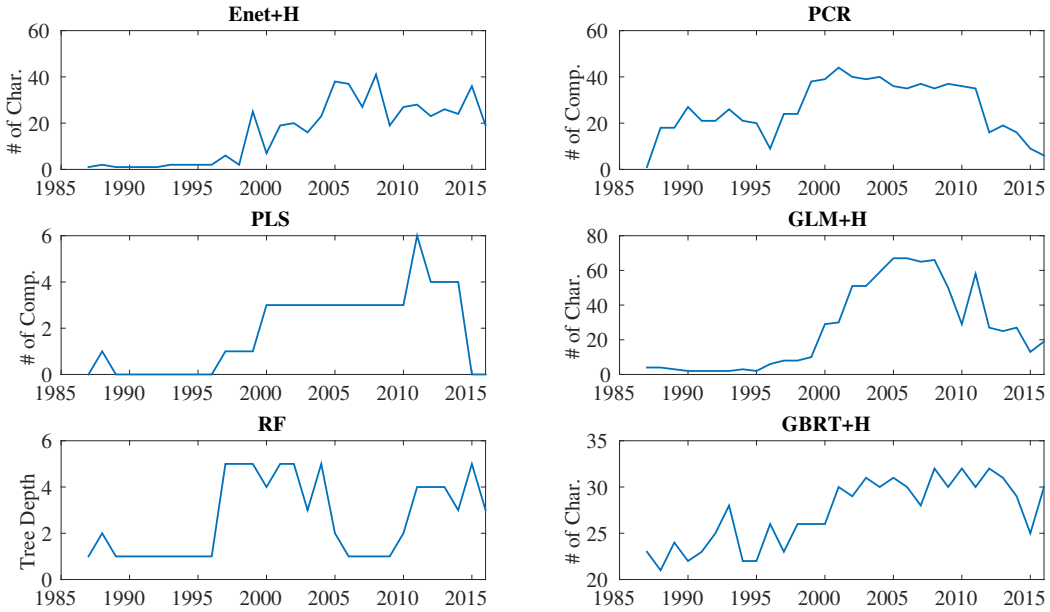


Note: In this table, we report R^2_{oos} for the entire panel of stocks using OLS with all variables (OLS), OLS using only size, book-to-market, and momentum (OLS-3), PLS, PCR, elastic net (ENet), generalize linear model (GLM), random forest (RF), gradient boosted regression trees (GBRT), and neural networks with one to five layers (NN1–NN5). "+H" indicates the use of Huber loss instead of the l_2 loss. We also report these R^2_{oos} within subsamples that include only the top 1,000 stocks or bottom 1,000 stocks by market value. The lower panel provides a visual comparison of the R^2_{oos} statistics in the table (omitting OLS model due to its large negative values).

(which pre-selects size, book-to-market, and momentum as the only covariates), PLS, PCR, elastic net (ENet), generalized linear model with group lasso (GLM), random forest (RF), gradient boosted regression trees (GBRT), and neural networks architectures with one to five layers (NN1,...,NN5). For OLS, ENet, GLM, and GBRT, we present their robust versions using Huber loss, which perform better than the version without.

The first row of Table 1 reports R^2_{oos} for the entire pooled sample. The OLS model using all 920 features produces an R^2_{oos} of -4.60%, indicating it is handily dominated by a naive forecast of zero excess return for all stocks in all months. This may be unsurprising as the lack of regularization leaves OLS highly susceptible to in-sample overfit. However, restricting OLS to a sparse parameterization, either by forcing the model to include only three covariates (size, value, and momentum), or by penalizing the specification with the elastic net—generates a substantial improvement over the full OLS model (R^2_{oos} of 0.16% and 0.09% respectively). Figure 3 summarizes model complexity for each model at each re-estimation date. The upper left panel shows the number of features to which elastic

Figure 3: Time-varying Model Complexity



Note: This figure demonstrates the model complexity for elastic net (ENet), PCR, PLS, generalized linear model with group lasso (GLM), random forest (RF) and gradient boosted regression trees (GBRT) in each training sample of our 30-year recursive out-of-sample analysis. For ENet and GLM we report the number of features selected to have non-zero coefficients; for PCR and PLS we report the number of selected components; for RF we report the average tree depth; and for GBRT we report the number of distinct characteristics entering into the trees.

net assigns a non-zero loading. In the first ten years of the test sample, the model typically chooses fewer than five features. After 2000, the number of selected features rises and hovers between 20 and 40 (we discuss the identities of leading predictors in the next subsection).

Regularizing the linear model via dimension reduction improves predictions even further. By forming a few linear combinations of predictors, PLS and especially PCR, raise the out-of-sample R^2 to 0.18% and 0.28%, respectively. Figure 3 shows that PCR typically uses 20 to 40 features in its forecasts. PLS, on the other hand, fails to find a single reliable component for much of the early sample, but eventually settles on three to six components. The improvement of dimension reduction over variable selection and shrinkage via elastic net suggests that our characteristic-based features can be thought of as partially redundant and fundamentally noisy signals, and combining them into a low-dimension components improves their value by averaging-out noise to better reveal their correlated signals.

The generalized linear model with group lasso penalty fails to improve on the performance of purely linear methods (R^2_{oos} of 0.19%). The fact that this method uses spline functions of individual features, but includes no interaction among features, suggesting that univariate expansions provide little incremental information beyond the linear model. Though it tends to select more features than

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

	OLS-3 +H	PLS	PCR	ENet +H	GLM +H	RF	GBRT +H	NN1	NN2	NN3	NN4	NN5
OLS+H	**3.81**	**3.82**	**3.85**	**3.81**	**3.83**	**3.91**	**3.94**	**3.96**	**3.96**	**3.98**	**3.97**	**3.96**
OLS-3+H		0.23	1.72	-0.80	0.63	1.55	1.93	**1.98**	**2.83**	**3.01**	**2.61**	**2.63**
PLS			1.58	-0.71	0.08	1.39	1.61	1.52	**2.29**	**2.43**	**2.18**	**2.15**
PCR				-1.51	-1.62	0.06	0.48	0.54	1.13	1.20	0.94	0.85
ENet+H					1.00	1.59	1.79	**2.09**	**2.02**	**2.19**	1.92	1.94
GLM+H						1.21	1.59	1.70	**2.55**	**2.76**	**2.44**	**2.33**
RF							0.66	0.66	1.12	1.30	0.94	0.90
GBRT+H								0.24	0.73	0.83	0.53	0.46
NN1									0.87	1.11	0.49	0.31
NN2										0.10	-1.09	-1.20
NN3											-1.03	-1.92
NN4												-0.47

Note: This table reports pairwise Diebold-Mariano test statistics comparing the out-of-sample stock-level prediction performance among thirteen models. Positive numbers indicate the column model outperforms the row model. Bold font indicates the difference is significant at 5% level or better.

elastic net, those additional features do not translate into incremental performance.

Boosted trees and random forests are competitive with PCR, producing fits of 0.27% and 0.30%, respectively. Random forests generally estimates shallow trees, with one to five layers on average. To quantify the complexity of GBRT, we report the number of features used in the boosted tree ensemble at each re-estimation point. In the beginning of the sample GBRT uses around 20 features to partition outcomes, with this number increasing to 30 later in the sample. In terms of complexity, all methods exhibit the same pattern that the number of reliable features increases over time.[24]

Neural networks are the best performing nonlinear method, and the best predictor overall, with an $R^2_{\text{oos}} = 0.35\%$ for NN1 and peaking at 0.39% for NN3. These results point to the value of incorporating complex predictor interactions, which are embedded in tree and neural network models but that are missed by other techniques. The results also show that in the monthly return setting, the benefits of "deep" learning are limiting, as four and five layer models fail to improve over NN3.

The second and third rows of Table 1 break out predictability for large stocks (the top 1,000 stocks by market equity each month) and small stocks (the bottom 1,000 each month). These are based on the full estimated model (using all stocks), but focuses on fits among the two subsamples. The baseline pattern that OLS fares poorly, regularized linear models are an improvement, and nonlinear models dominate carries over into these subsamples. Tree methods and neural nets are especially successful among large stocks, with R^2_{oos}'s ranging from 0.53% to 0.72%. This dichotomy provides reassurance that machine learning is not merely picking up small scale inefficiencies driven by illiquidity.

As an aside, it is useful to know that there is a roughly 3% inflation in out-of-sample R^2s if performance is benchmarked against historical averages. For OLS-3, the R^2 relative to the historical

[24] Because we hold the five neural nets architectures fixed and simply compare across them, we do not describe their estimated complexity in Figure 3.

mean forecast is 3.74% per month! Evidently, the historical mean is such a noisy forecaster that it is easily beaten by a fixed excess return forecasts of zero.

While Table 1 offers a quantitative comparison of models' predictive performance, Table 2 assesses the statistical significance of differences among models. It reports Diebold-Mariano test statistics for pairwise comparisons of a column model versus a row model. Diebold-Mariano statistics are distributed $N(0,1)$ under the null of no difference between models, thus the test statistic magnitudes map to p-values in the same way as regression t-statistics. Our sign convention is that a positive statistic indicates the column model outperforms the row model.

The first conclusion from Table 2 is that the performance differences among regularized linear models are all insignificant. Restricting OLS to only use three predictors, reducing dimension via PLS or PCA, and penalizing via elastic-net produce statistically indistinguishable forecast performance. They are also indistinguishable from the generalized linear model with group lasso penalty and random forest. Boosted trees, on the other hand, deliver a (marginally) significant improvements over linear models. Next, neural networks generally produce large and significant statistical improvements over linear and generalized linear models, but their improvements over tree models are statistically insignificant.

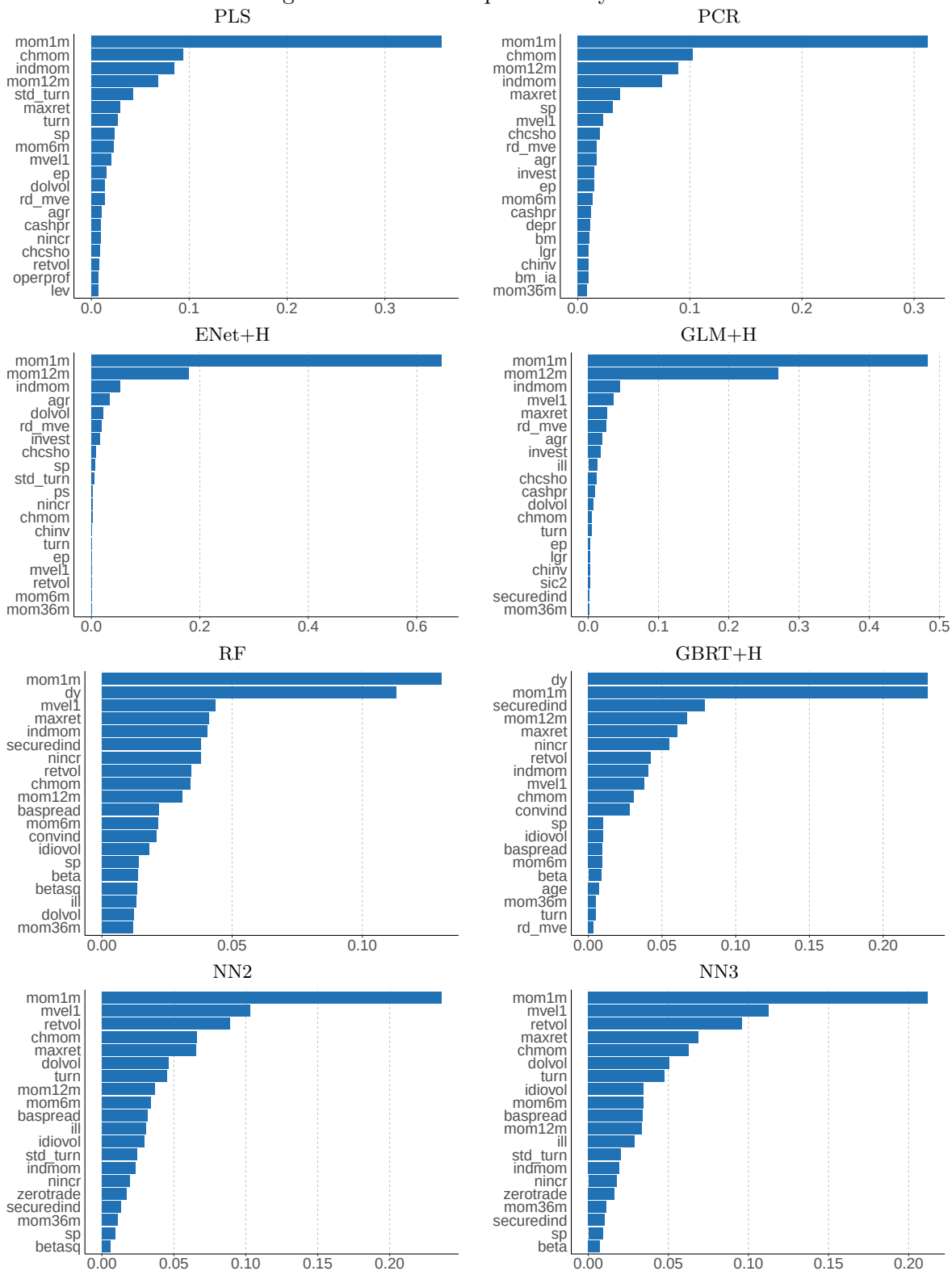
3.3 Which Covariates Matter?

We now investigate the relative importance of individual covariates for the performance of each model. First, for each method, we calculate variable importance for stock-level characteristics according to Section 2.9 within each training sample of our recursive out-of-sample procedure, and average these into a single importance measure for each predictor. Figure 4 reports the variable importance for the top 20 stock-level characteristics for each method. Variable importances within a given model are normalized to sum to one, giving them the interpretation of relative importance for that particular model.

We also translate the importance of each characteristic into the corresponding ranks for each method, then sum their ranks over all methods. In Figure 5, characteristics with the highest total ranks are placed on top, and the color gradient within each column shows the model-specific ranking of characteristics from least to most important (lightest to darkest).

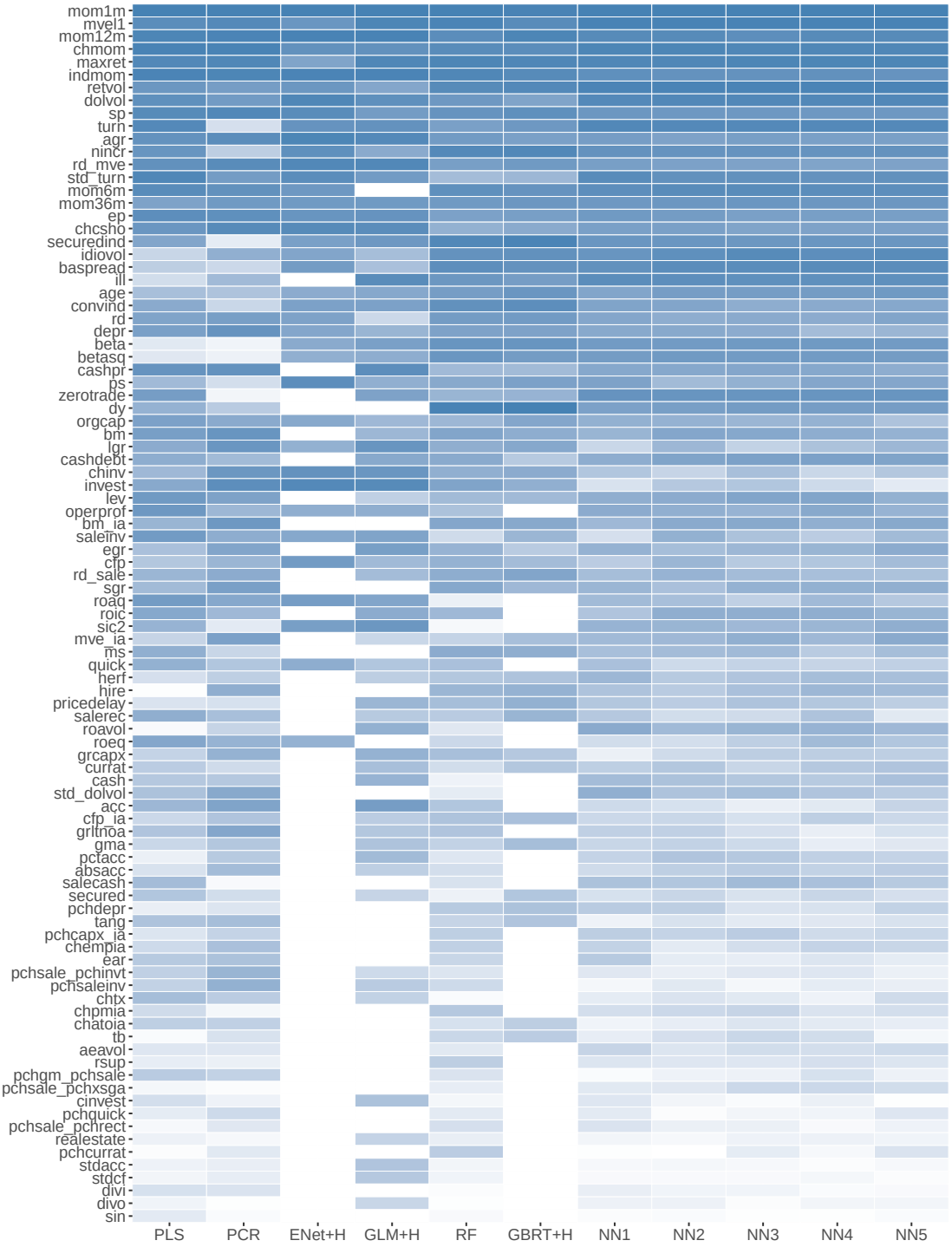
Figures 4 and 5 demonstrate that models are generally in close agreement regarding the most influential stock-level predictors, which can be grouped into four categories. The first are based on recent price trends, including five of the top seven variables in Figure 5: short-term reversal (mom1m), stock momentum (mom12m), momentum change (chmom), industry momentum (indmom), recent maximum return (maxret), and long-term reversal (mom36m). Next are liquidity variables, including turnover and turnover volatility (turn, std_turn), log market equity (mvel1), dollar volume (dolvol), Amihud illiquidity (ill), number of zero trading days (zerotrade), and bid-ask spread (baspread). Risk measures constitute the third influential group, including total and idiosyncratic return volatility (retvol, idiovol), market beta (beta), and beta-squared (betasq). The last group includes valuation ratios and fundamental signals, such as earnings-to-price (ep), sales-to-price (sp), asset growth (agr), and number of recent earnings increases (nincr).

Figure 4: Variable Importance By Model



Note: Variable importance for the top 20 most influential variables in each model. We recalculate variable importance within each of our 30 recursing training samples and then average these values. Finally, the variable importances within a given model are normalized to sum to one and are thus interpretable as relative importance within a given model.

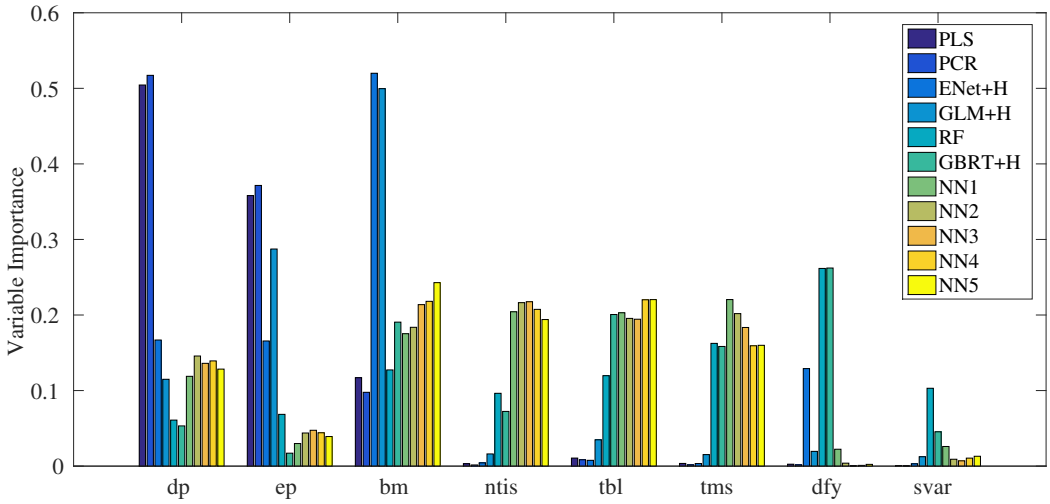
Figure 5: Characteristic Importance



Note: This figure describes how each model ranks the 94 stock-level characteristics in terms of overall model contribution. Columns correspond to individual models, and color gradients within each column indicate the most influential (dark blue) to least influential (white) variables. Characteristics are ordered based on the sum of their ranks over all models, with the most influential characteristics on top and least influential on bottom.

Table 3: Variable Importance for Macroeconomic Predictors

	PLS	PCR	ENet+H	GLM+H	RF	GBRT+H	NN1	NN2	NN3	NN4	NN5
dp	50.44	51.72	16.69	11.49	6.09	5.32	11.89	14.57	13.60	13.92	12.84
ep	35.80	37.15	16.56	28.73	6.85	1.71	2.99	4.38	4.74	4.41	3.91
bm	11.70	9.77	52.00	49.96	12.73	19.06	17.52	18.37	21.37	21.81	24.28
ntis	0.32	0.16	0.43	1.61	9.64	7.24	20.43	21.63	21.76	20.75	19.40
tbl	1.06	0.84	0.77	3.48	11.97	20.06	20.30	19.55	19.44	22.02	22.04
tms	0.36	0.17	0.34	1.52	16.25	15.84	22.04	20.18	18.35	15.93	15.99
dfy	0.25	0.17	12.90	1.95	26.16	26.22	2.24	0.39	0.03	0.10	0.23
svar	0.06	0.03	0.31	1.24	10.31	4.54	2.60	0.91	0.70	1.06	1.30



Note: This table reports the average importance of eight macroeconomic variables in each model. We recalculate variable importance within each of our 30 recursing training samples and then average these values. Finally, the variable importances within a given model are normalized to sum to one and are thus interpretable as relative importance within a given model. The lower panel provides a complementary visual comparison of macroeconomic variable importances.

Figure 4 shows that characteristic importance magnitudes for penalized linear models (elastic net and generalized linear model) and dimension reduction models are highly skewed toward momentum and reversal. Trees and neural networks are more democratic, drawing predictive information from a broader set of characteristics.

Table 3 shows the importance of macroeconomic predictor variables (again normalized to sum to one within a given model). An interesting distinction emerges between linear and nonlinear models. All methods agree that the aggregate book-to-market ratio is a critical predictor. Linear models (including PLS, PCR, elastic net, and generalized linear) also strongly favor dividend-to-price, earnings-to-price, and place almost zero weight on bond market variables and equity issuance. Nonlinear methods (trees and neural networks) place great emphasis on exactly those predictors ignored by linear methods—including Treasury rates, term spreads, default spreads, and issuance activity. Finally, market volatility has little role in any model.

3.4 Portfolio Forecasts

3.4.1 Pre-specified Portfolios

Table 4: Portfolio-level Out-of-Sample Predictive R^2

	OLS-3 +H	PLS	PCR	ENet +H	GLM +H	RF	GBRT +H	NN1	NN2	NN3	NN4	NN5
S&P 500	-0.11	-0.86	-2.62	-0.38	0.86	1.39	1.13	0.84	0.96	1.80	1.46	1.60
Big Growth	0.41	0.75	-0.77	-1.55	0.73	0.99	0.80	0.70	0.32	1.67	1.42	1.40
Big Value	-1.05	-1.88	-3.14	-0.03	0.70	1.41	1.04	0.78	1.20	1.57	1.17	1.42
Big Neutral b/m	0.12	-0.81	-2.39	-0.46	0.41	1.05	1.03	1.33	0.78	1.81	1.93	1.93
Small Growth	0.35	1.54	0.72	-0.03	0.95	0.54	0.62	1.68	1.26	1.48	1.53	1.44
Small Value	-0.06	0.40	-0.12	-0.57	0.02	0.71	0.90	0.00	0.47	0.46	0.41	0.53
Small Neutral b/m	-0.01	0.78	-0.10	-0.25	0.36	0.41	0.38	0.58	0.55	0.68	0.62	0.72
Big Conservative	-0.24	-0.17	-1.97	0.19	0.69	0.96	0.78	1.08	0.67	1.68	1.46	1.56
Big Aggressive	-0.12	-0.77	-2.00	-0.91	0.68	1.83	1.45	1.14	1.65	1.87	1.55	1.69
Big Neutral inv	-0.36	-1.65	-3.20	-0.11	0.76	0.99	0.73	0.54	0.62	1.62	1.44	1.60
Small Conservative	0.02	0.75	0.48	-0.46	0.55	0.59	0.60	0.94	0.91	0.93	0.99	0.88
Small Aggressive	0.14	0.97	0.06	-0.54	0.19	0.86	1.04	0.25	0.66	0.75	0.67	0.79
Small Neutral inv	-0.04	0.53	-0.17	0.08	0.45	0.23	0.20	0.73	0.60	0.81	0.73	0.80
Big Robust	-0.58	-0.22	-2.89	-0.27	1.54	1.41	0.70	0.60	0.84	1.14	1.05	1.21
Big Neutral op	-0.24	-1.47	-1.95	-0.40	-0.26	0.67	0.83	0.24	0.60	1.21	0.95	1.07
Big Weak	-0.08	-1.02	-2.77	-0.21	0.10	1.46	1.44	0.95	1.00	1.78	1.70	1.73
Small Robust	-0.77	0.77	0.18	-0.32	0.41	0.27	-0.06	-0.06	-0.02	0.06	0.13	0.15
Small Weak	0.02	0.32	-0.28	-0.25	0.17	0.90	1.31	0.84	0.85	1.09	0.96	1.08
Small Neutral op	0.22	1.05	0.09	0.03	0.48	0.76	0.97	1.08	1.04	1.19	1.12	1.18
Big Up	-1.53	-2.54	-3.93	-0.21	0.40	1.12	0.68	0.46	0.85	1.28	0.99	1.05
Big Down	-0.10	-1.20	-2.05	-0.26	0.36	1.09	0.77	0.48	0.89	1.34	1.17	1.36
Big Medium	0.24	1.38	0.57	0.01	1.32	1.56	1.37	1.60	1.76	2.28	1.83	2.01
Small Up	-0.79	0.42	-0.36	-0.33	-0.33	0.31	0.40	0.23	0.60	0.67	0.55	0.61
Small Down	0.40	1.16	0.47	-0.46	0.62	0.93	1.20	0.80	0.97	0.97	0.97	0.96
Small Medium	-0.29	0.03	-0.61	-0.56	-0.20	0.11	0.18	0.05	0.29	0.41	0.30	0.45

Note: In this table, we report the out-of-sample predictive R^2s for 25 portfolios using OLS with size, book-to-market, and momentum, OLS-3, PLS, PCR, elastic net (ENet), generalized linear model with group lasso (GLM), random forest (RF), gradient boosted regression trees (GBRT), and five architectures of neural networks (NN1,...,NN5), 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 the Fama-French value, investment, and profitability factors, as well as momentum.

So far we have analyzed predictability of individual stock returns. Next, we compare forecasting performance of machine learning methods for aggregate portfolio returns. We build bottom-up forecasts by aggregating individual stock return predictions into portfolios based on stocks' portfolio weights. Given the weight of stock i in portfolio p, $w_{i,t}^p$, and given a model-based out-of-sample forecast for stock i, $\widehat{r}_{i,t+1}$, we construct the portfolio return forecast as

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

This bottom-up approach works for any target portfolio whose weights are known a priori.

Because the model is optimized for stock-level forecasts, studying portfolio forecasts provides an additional indirect evaluation of the model and its robustness. First, aggregate portfolios tend to be of broader economic interest because they represent the risky-asset savings vehicles most commonly held by investors (via mutual funds, ETFs, and hedge funds). And, by studying value-weighted portfolios, we can assess the extent to which a model's predictive performance thrives in the most valuable (and most economically important) assets in the economy. Second, the distribution of portfolio returns is sensitive to dependence among stock returns, with the implication that a good stock-level prediction model is not guaranteed to produce accurate portfolio-level forecasts. Bottom-up portfolio forecasts allow us to evaluate a model's ability to transport its asset predictions, which occur at the finest asset level, into broader investment contexts.

We form machine learning bottom-up forecasts for 25 of the most well known portfolios in the empirical finance literature, including the S&P 500, six size and value portfolios, six size and investment portfolios, six size and profitability portfolios, and six size and momentum portfolios. In all cases, we create the portfolios ourselves using CRSP market equity value weights. Our portfolio construction differs slightly from the actual S&P 500 index and the characteristic-based Fama-French portfolios (Fama and French, 1993, 2015), but has the virtue that we can exactly track the ex ante portfolio weights of each.[25]

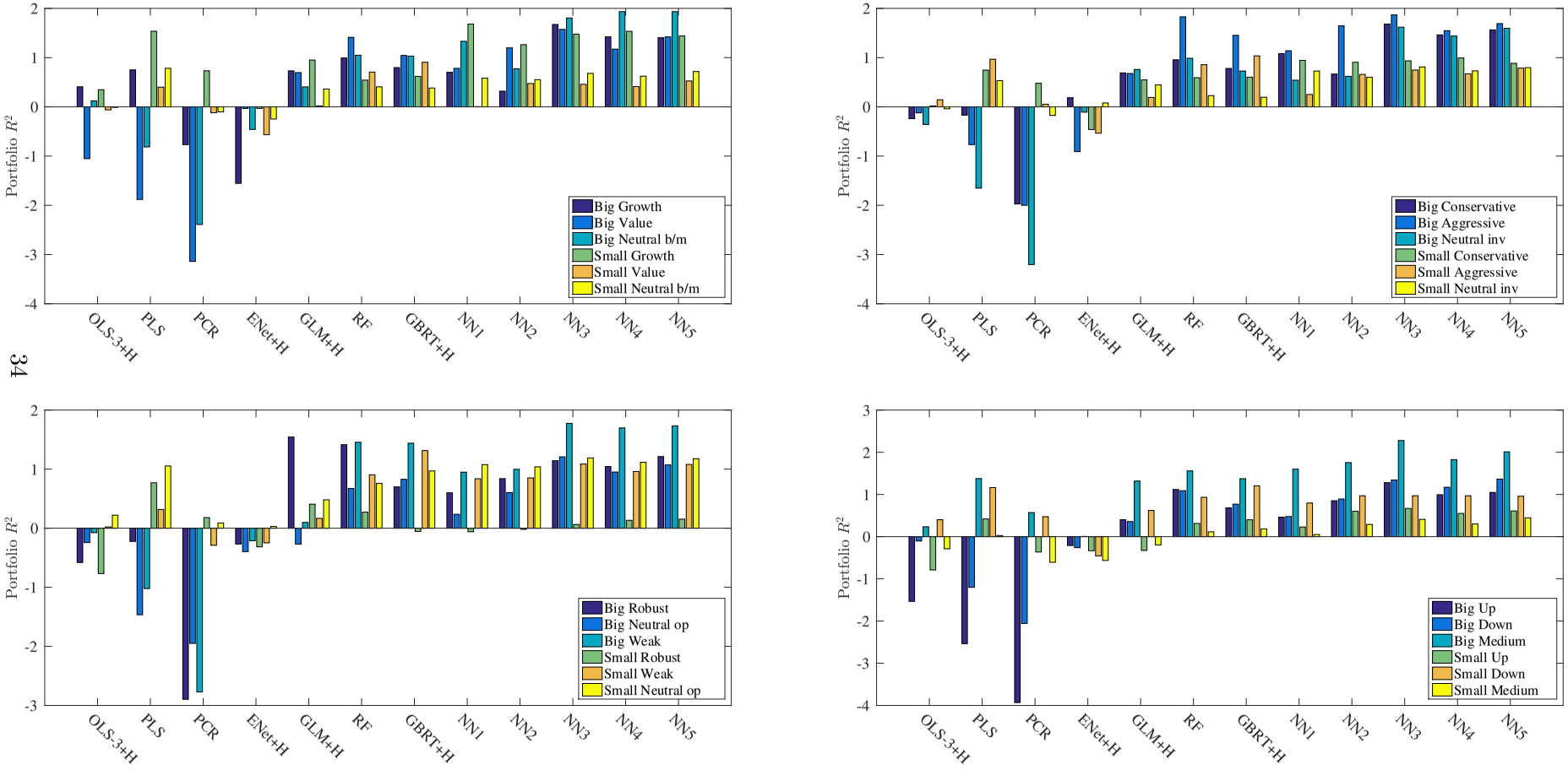
Table 4 provides the out-of-sample R^2s over our 30-year testing sample (Figure 6 provides a corresponding graphical illustration). The first interesting finding is that regularized linear methods fail to outperform naive constant forecasts of the S&P 500, while all nonlinear models have substantial positive predictive performance. The one month out-of-sample R^2 is 0.86% for the generalized linear model and reaches as high as 1.80% for the three-layer neural network. As a benchmark for comparison, nearly all of the macroeconomic return predictor variables in the survey of Welch and Goyal (2008) fail to produce a positive out-of-sample forecast R^2. Kelly and Pruitt (2013) find that PLS delivers out-of-sample forecasting R^2's around 1% per month for the aggregate market index, though their forecasts directly target the market return as opposed to being bottom-up forecasts. And the most well-studied portfolio predictors, such as the aggregate price-dividend ratio, typically produce an *in-sample* predictive R^2 of around 1% per month Cochrane (2007).[26] In short, machine learning methods, and nonlinear methods in particular, produce unusually powerful out-of-sample predictions of the market index.

The patterns in S&P 500 forecasting performance across models carry over to characteristic-based long-short portfolios. Nonlinear methods excel—random forests and NN3–NN5 produce positive R^2's for *every* portfolio analyzed. Boosted trees, NN1, and NN2 R^2's are positive for all but one portfolio. The generalized linear model R^2 is positive for 22 out of 25 portfolios. Linear methods, on the other hand, are mixed. They tend to produce negative R^2's for large stocks and are marginally positive for

[25]Our version of the S&P 500 is weighted by market equity value, which differs slightly from its actual float-weighted construction. And our Fama-French portfolio sorts differ due to minor differences in data filtering, with average correlation as high as 97.9% across all portfolios.

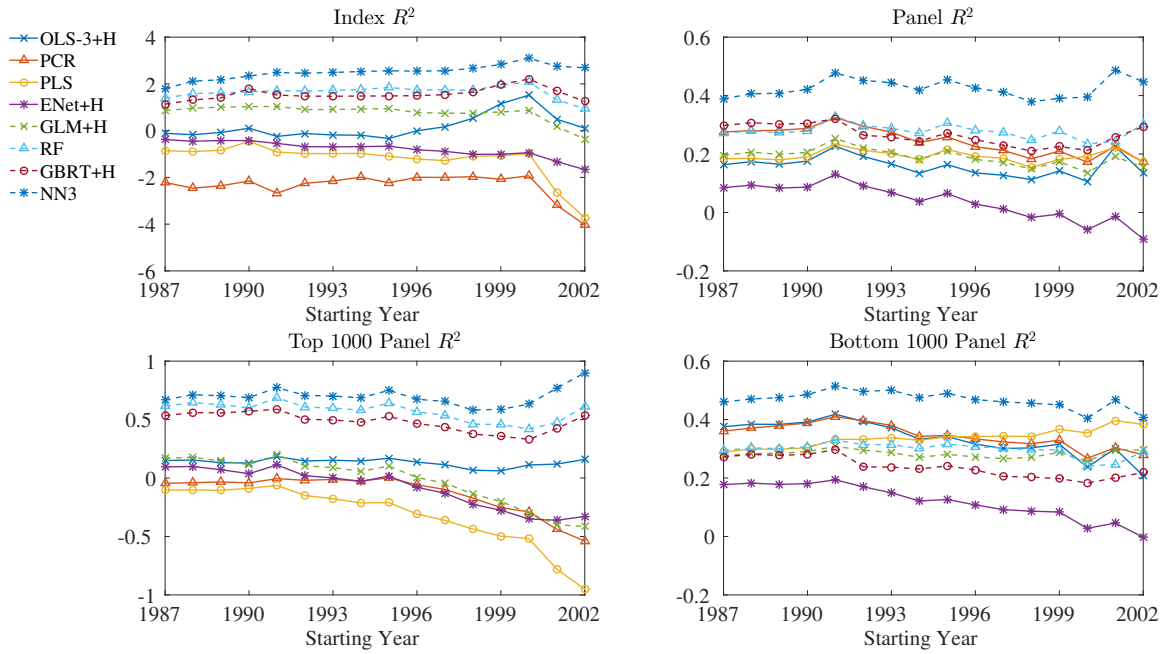
[26]While these predictive R^2's seem small from an explained variance standpoint, 1% at the monthly frequency translates into large investor utility gains. See, e.g., Campbell and Thompson (2008) and Cochrane (2007).

Figure 6: Portfolio-level Out-of-Sample Predictive R^2



Note: Graphical illustration of portfolio-level R^2 for the double-sorted portfolios in Table 4.

Figure 7: R^2_{oos} by Test Sample Starting Year



Note: We vary the starting year of our out-of-sample testing period, holding the ending fixed at 2016. E.g., points corresponding to tick-mark 2002 are based on the 15 year out-of-sample window from 2002 to 2016. The four figures report the market index R^2, panel R^2, Top 1000 panel R^2, Bottom panel R^2 of each model for different starting years.

small stocks but, on balance, regularized linear models appear unsuccessful in bottom-up portfolio return forecasting.

The out-of-sample S&P 500 forecasting performance in Table 4 is based on a 30-year testing sample. Kelly and Pruitt (2013) point out that out-of-sample R^2's can be sensitive to choice of testing sample. The upper left panel of Figure 7 plots out-of-sample predictive R^2 for S&P 500 index returns based on a wide range of testing sample. Our findings are not specific to a particular test sample—for all sample splits, NN3 remains the dominant model, and the relative rankings of other methods is roughly preserved as well. The robustness of our findings to sample split also holds for our stock-level analyses. The upper right panel of Figure 7 shows that our findings in terms of stock panel R^2 is robust to choice of testing sample, and the lower panels show that this remains true among sub-cohorts of large and small stocks.

3.4.2 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 one-month-ahead out-of-sample stock return predictions for each method. We then sort stocks into deciles based on models' forecasts. We reconstitute portfolios each month using equal weights, which are the most natural choices given that our objective functions minimize an equal weighted average of squared

Table 5: Performance of Machine Learning Portfolios

	OLS-3+H				PLS				PCR			
	Pred	Avg	Std	SR	Pred	Avg	Std	SR	Pred	Avg	Std	SR
Low(L)	-0.41	0.19	6.52	0.10	-0.85	-0.05	6.73	-0.03	-0.92	-0.49	7.02	-0.24
2	-0.08	0.40	5.28	0.26	-0.26	0.31	6.02	0.18	-0.29	0.15	6.20	0.08
3	0.15	0.59	4.82	0.42	0.04	0.46	5.66	0.28	0.03	0.38	5.51	0.24
4	0.37	0.72	4.45	0.56	0.29	0.64	5.32	0.41	0.27	0.54	5.24	0.35
5	0.57	0.77	4.49	0.60	0.51	0.70	5.19	0.47	0.49	0.65	5.06	0.44
6	0.78	0.67	4.66	0.49	0.71	0.77	5.14	0.52	0.69	0.72	4.96	0.51
7	0.98	0.64	5.15	0.43	0.92	0.78	5.11	0.53	0.89	0.91	5.14	0.61
8	1.20	0.64	5.98	0.37	1.17	0.87	5.23	0.58	1.13	1.13	5.17	0.75
9	1.44	0.76	7.10	0.37	1.50	1.10	5.34	0.71	1.45	1.34	5.44	0.85
High(H)	1.81	1.84	8.52	0.75	2.15	1.51	5.79	0.90	2.09	1.91	6.08	1.09
H-L	2.22	1.65	6.41	0.89	3.01	1.56	4.45	1.22	3.01	2.40	4.65	1.79
	ENet+H				GLM+H				RF			
	Pred	Avg	Std	SR	Pred	Avg	Std	SR	Pred	Avg	Std	SR
Low(L)	0.06	-0.31	7.01	-0.15	-0.43	-0.51	6.94	-0.25	0.26	-0.33	7.13	-0.16
2	0.33	0.39	6.03	0.22	0.02	0.33	6.08	0.19	0.41	0.32	5.68	0.19
3	0.50	0.62	5.13	0.42	0.28	0.54	5.59	0.33	0.49	0.52	5.31	0.34
4	0.63	0.63	4.83	0.45	0.49	0.64	5.40	0.41	0.56	0.60	5.21	0.40
5	0.76	0.71	4.77	0.52	0.66	0.71	5.20	0.47	0.62	0.63	5.14	0.42
6	0.89	0.75	4.97	0.52	0.81	0.79	4.97	0.55	0.68	0.73	5.10	0.50
7	1.02	0.78	5.26	0.51	0.97	0.82	4.92	0.58	0.75	0.85	5.19	0.57
8	1.16	0.85	5.51	0.53	1.14	0.99	5.09	0.67	0.81	0.90	5.33	0.58
9	1.34	1.03	5.92	0.60	1.37	1.29	5.59	0.80	0.89	1.17	5.64	0.72
High(H)	1.65	1.80	7.31	0.86	1.84	1.64	6.34	0.90	1.07	1.83	6.78	0.93
H-L	1.59	2.12	5.48	1.34	2.27	2.15	4.39	1.70	0.81	2.16	5.34	1.40
	GBRT+H				NN1				NN2			
	Pred	Avg	Std	SR	Pred	Avg	Std	SR	Pred	Avg	Std	SR
Low(L)	-0.03	-0.38	6.67	-0.20	-0.47	-0.80	7.47	-0.37	-0.37	-0.82	7.97	-0.36
2	0.16	0.43	5.66	0.26	0.14	0.21	6.24	0.12	0.19	0.17	6.44	0.09
3	0.27	0.55	5.45	0.35	0.43	0.45	5.58	0.28	0.43	0.43	5.48	0.27
4	0.36	0.77	5.34	0.50	0.64	0.64	5.05	0.44	0.59	0.65	4.86	0.46
5	0.46	0.68	5.28	0.45	0.80	0.73	4.75	0.53	0.73	0.72	4.59	0.54
6	0.54	0.78	5.12	0.53	0.96	0.85	4.69	0.63	0.85	0.81	4.50	0.62
7	0.62	0.65	5.19	0.44	1.12	0.82	4.72	0.60	0.98	0.86	4.55	0.65
8	0.71	0.94	5.27	0.61	1.33	0.97	4.94	0.68	1.13	0.92	4.82	0.66
9	0.81	1.11	5.26	0.73	1.64	1.21	5.49	0.76	1.40	1.13	5.46	0.72
High(H)	1.02	1.70	6.57	0.90	2.46	2.13	7.30	1.01	2.32	2.36	8.03	1.02
H-L	1.04	2.08	4.25	1.70	2.93	2.93	4.81	2.11	2.69	3.18	4.90	2.25
	NN3				NN4				NN5			
	Pred	Avg	Std	SR	Pred	Avg	Std	SR	Pred	Avg	Std	SR
Low(L)	-0.39	-0.96	7.77	-0.43	-0.28	-0.90	7.87	-0.40	-0.21	-0.76	7.93	-0.33
2	0.17	0.13	6.42	0.07	0.25	0.18	6.57	0.09	0.25	0.24	6.58	0.13
3	0.42	0.53	5.47	0.34	0.47	0.46	5.60	0.28	0.46	0.51	5.67	0.31
4	0.59	0.68	4.90	0.48	0.61	0.59	4.91	0.41	0.59	0.61	5.07	0.42
5	0.73	0.77	4.70	0.57	0.72	0.66	4.55	0.50	0.70	0.65	4.58	0.49
6	0.86	0.81	4.54	0.62	0.82	0.77	4.45	0.60	0.80	0.77	4.43	0.60
7	0.99	0.90	4.58	0.68	0.92	0.86	4.51	0.66	0.90	0.85	4.60	0.64
8	1.16	0.97	4.85	0.69	1.07	1.05	4.80	0.76	1.04	0.91	4.88	0.65
9	1.44	1.16	5.50	0.73	1.32	1.22	5.60	0.75	1.30	1.24	5.54	0.77
High(H)	2.30	2.23	7.78	0.99	2.28	2.35	7.95	1.02	2.19	2.21	7.78	0.98
H-L	2.69	3.19	4.77	2.32	2.56	3.25	4.79	2.35	2.39	2.97	5.05	2.03

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. 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.

Table 6: Drawdowns, Turnover, and Subsamples of Machine Learning Portfolios

	OLS-3 +H	PLS	PCR	ENet +H	GLM +H	RF	GBRT +H	NN1	NN2	NN3	NN4	NN5
Max DD(%)	53.89	25.19	32.20	32.19	25.58	44.07	30.03	19.18	21.12	14.39	17.72	19.76
Max 1M Loss(%)	32.09	20.14	19.93	28.91	15.57	29.49	20.44	15.61	21.12	9.65	17.02	19.51
Turnover(%)	75.30	44.25	50.54	69.18	62.11	53.88	65.94	51.73	52.13	52.48	53.02	52.81
Expansions:												
AvgRet	1.51	1.47	2.42	2.13	2.08	2.04	2.09	2.85	3.04	3.09	2.99	2.80
Std	6.37	4.26	4.55	5.24	4.06	4.97	4.03	4.50	4.55	4.49	4.69	4.77
SR	0.82	1.20	1.84	1.40	1.77	1.42	1.79	2.19	2.32	2.39	2.21	2.03
Recessions:												
AvgRet	2.93	2.41	2.46	2.04	2.85	3.38	2.08	3.71	4.61	4.95	4.81	4.59
Std	6.57	5.88	6.20	7.37	6.74	7.98	5.98	7.13	6.34	6.29	6.91	7.01
SR	1.54	1.42	1.37	0.96	1.47	1.47	1.21	1.80	2.52	2.72	2.41	2.27

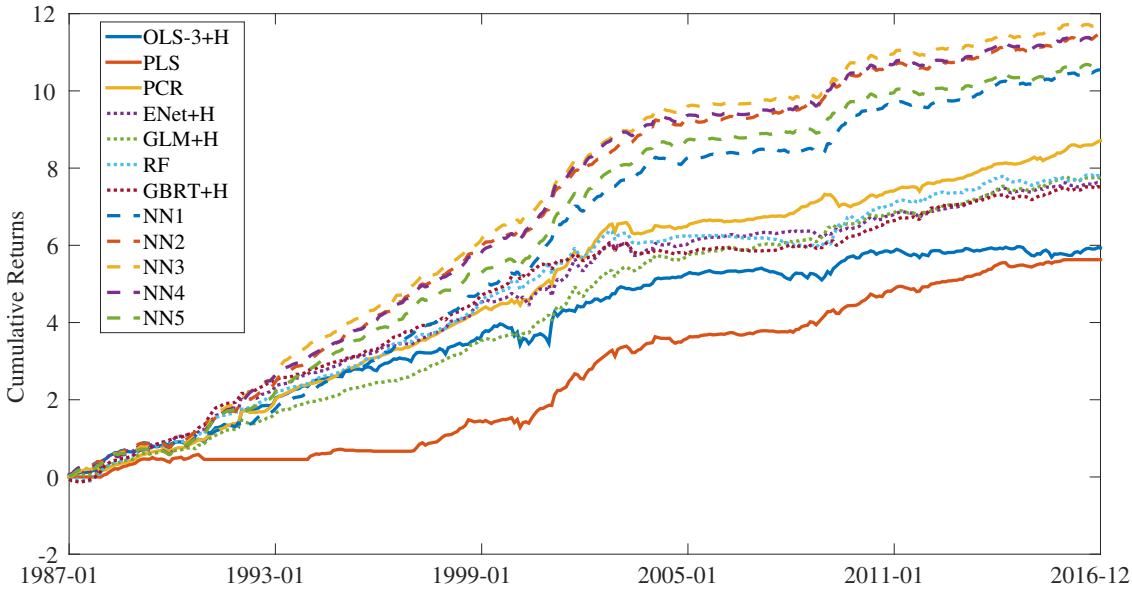
Note: In this table, we report more empirical results of prediction-sorted portfolios over the 30-year out-of-sample testing period. Max DD reports the maximum drawdown of each strategy in over our sample and is calculated as $\text{MaxDD}(T) = \max_{0 \le t_1 \le t_2 \le T} (Y_{t_1} - Y_{t_2})$. Max 1M Loss is the most extreme negative monthly return of each strategy over our sample. Turnover describes the average monthly percentage change in total holdings, defined as $\text{Turnover}(T) = \frac{1}{T}\sum_{t=1}^{T}\left(\sum_{i \in B_t/B_{t-1}} w_{i,t} + \sum_{i \in S_t/S_{t-1}} w_{i,t}\right)$, where $w_{i,t}$ is the weight of stock i in the portfolio at time t, and B_t represents long positions and S_t represents short positions at time t within the prediction-sorted portfolios. We also report the average return(AvgRet), standard deviation(Std), sharp ratio(SR) of portfolios during expansions (326 months) and recessions (34 months) according to NBER recession dates. and recession time. We use NBER recession dummy to divide 360 OOS months into two partitions. Good time contains 326 months and recession time contains 34 months.

forecast errors. Finally, we construct a zero-net-investment portfolio that buys the highest expected return stocks (decile 10) and sells the lowest (decile 1).

Table 5 reports results. Out-of-sample portfolio performance aligns very closely with machine learning forecasts. Realized returns increase monotonically with machine learning forecasts from every method (with minor exceptions, such as decile 7 of NN1). The neural network models again dominate linear models and tree-based approaches. In particular, for all but the most extreme deciles, the quantitative match between predicted returns and average realized returns using neural networks is extraordinarily close. The best 10–1 portfolio returns on average 3.25% per month using NN4. Its monthly volatility is 4.79% (16.6% on an annualized basis), amounting to an annualized out-of-sample Sharpe ratio of 2.35.

Table 6 reports maximum drawdowns, portfolio turnover, and expansion/recession performance of 10-1 spread portfolios based on return forecasts from each method. Not only do neural networks portfolios have higher Sharpe ratios than alternatives, they also have much smaller drawdowns and less turnover than other approaches. The maximum drawdown experienced for the NN3 strategy is 14.39%, versus 53.89% maximum drawdown for OLS-3. Random forests and GBRT see drawdowns as large as 44.07% and 30.03%, respectively. Likewise, NN3's worst one month performance is a 9.65% loss, which represents the least extreme downside risk among all models. Finally, the cumulative performance of all strategies is plotted in Figure 8.

Figure 8: Cumulative Return of All Models



Note: We display the cumulative return of prediction-sorted portfolios for each method over the 30-year out-of-sample testing period.

4 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 neural networks and regression trees, and we isolate the source of their predictive advantage to accommodation of nonlinear 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 to 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, the most powerful predictors being associated with price trends and include return reversal and momentum. The next most powerful predictors are measures of stock liquidity, stock volatility, and valuation ratios.

The overall success of machine learning algorithms for return prediction brings promise for both economic modeling and for practical aspects of portfolio choice. With better measurement through machine learning, risk premia become less shrouded in estimation error, thus the challenge of iden-

tifying reliable economic mechanisms behind asset pricing phenomena becomes less steep. Finally, our findings help 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.

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.

Cochrane, John H, 2007, The dog that did not bark: A defense of return predictability, *The Review of Financial Studies* 21, 1533–1575.

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.

de Jong, Sijmen, 1993, Simpls: An alternative approach to partial least squares regression, *Chemometrics and Intelligent Laboratory Systems* 18, 251 – 263.

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

Dietterich, Thomas G, 2000, Ensemble methods in machine learning, in *International workshop on multiple classifier systems*, 1–15, Springer.

Eldan, Ronen, and Ohad Shamir, 2016, The power of depth for feedforward neural networks, in Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, eds., *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, 907–940 (PMLR, Columbia University, New York, New York, USA).

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.

Feng, Guanhao, Stefano Giglio, and Dacheng Xiu, 2017, Taming the factor zoo, Technical report, University of Chicago.

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.

Hansen, Lars Kai, and Peter Salamon, 1990, Neural network ensembles, *IEEE transactions on pattern analysis and machine intelligence* 12, 993–1001.

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).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, 2016, Deep residual learning for image recognition, in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 770–778.

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

Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh, 2006, A fast learning algorithm for deep belief nets, *Neural Comput.* 18, 1527–1554.

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, and Seth Pruitt, 2015, The three-pass regression filter: A new approach to forecasting using many predictors, *Journal of Econometrics* 186, 294–316.

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.

Masters, Timothy, 1993, *Practical Neural Network Recipes in C++* (Academic Press).

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.

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* .

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.

Internet Appendix

A 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 $t = 1, 2, \ldots, T$:

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

where c_t is an $N \times P_c$ matrix of characteristics, v_{t+1} is a 3×1 vector of factors, x_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 x_t from the following model:

$$x_t = \rho x_{t-1} + u_t,$$

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

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

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

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 $\text{sgn}(c_{i3,t} \times x_t)$. 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 $P_c = 50$, corresponding to $P = 200$ and 100, respectively, 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 PLS, PCR, Ridge, Lasso, Elastic Net (ENet), generalized

linear model with group lasso (GLM), random forest (RF), gradient boosted regression trees (GBRT), and the same five architectures of neural networks (NN1,...,NN5) 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 A.1: 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	7.75	2.14	8.61	0.52	3.37	-2.88	4.27	-4.61
PLS	6.50	2.97	6.30	2.19	0.97	-0.25	0.81	-0.17
PCR	2.25	0.92	1.28	0.44	0.49	-0.05	0.34	-0.04
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
GLM	6.15	3.85	6.18	3.84	3.48	1.04	3.44	1.00
GLM+H	6.10	3.86	6.12	3.85	3.40	1.06	3.34	1.03
RF	8.33	3.26	8.64	3.27	8.41	2.69	8.35	2.72
GBRT	7.13	3.23	7.34	3.27	6.32	2.63	6.13	2.61
GBRT+H	7.24	3.28	7.19	3.32	6.27	2.75	6.37	2.71
NN1	7.65	3.78	8.07	3.79	6.64	1.90	7.33	1.89
NN2	7.38	3.71	7.70	3.77	7.06	2.19	7.43	2.13
NN3	7.07	3.36	7.45	3.59	6.83	2.15	7.33	1.92
NN4	7.04	3.43	7.21	3.48	6.94	2.04	7.41	1.84
NN5	6.61	3.21	7.22	3.38	7.00	2.00	7.65	1.52
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), generalized linear model with group lasso (GLM), random forest (RF), gradient boosted regression trees (GBRT), and five architectures of neural networks (NN1,...,NN5), 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 A.1. 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 and almost identical 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. GLM is slightly better, but is dominated by NNs, RF, and GBRT. OLS is the worst in all settings, not surprisingly. PLS outperforms PCR in the linear model (a), but is dominated in the nonlinear case. When P_c increases, IS R^2s tend to increase whereas OOS R^2s decrease. Hence, the performance of all

methods deteriorates as overfitting exacerbates. Using Huber loss improves the OOS performance for all methods. RF, GBRT plus Huber loss remain the best choices for the nonlinear model. The comparison among NNs demonstrates a stark trade-off between model flexibility and implementation difficulty. Deeper models potentially allow for more parsimonious representation of the data, but their objective functions are more involved to optimize. For instance, the APG algorithm used in Elastic Net is not feasible for NNs, because its loss (as a function of weight parameters) is non-convex. As shown in the table, shallower NNs tend to outperform.

Table A.2: Comparison of Average Variable Selection Frequencies in Simulations

Model (a)

Parameter	Method	$c_{i1,t}$	$c_{i2,t}$	$c_{i3,t}$	$c_{i1,t} \times x_t$	$c_{i2,t} \times x_t$	$c_{i3,t} \times x_t$	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	0.52	0.59	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	0.54	0.59	0.92	0.08
	GLM	0.95	0.97	0.63	0.67	0.69	0.93	0.12
	GLM+H	0.95	0.97	0.62	0.68	0.67	0.93	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	0.58	0.51	0.56	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	0.58	0.53	0.56	0.92	0.05
	GLM	0.95	0.97	0.60	0.64	0.64	0.93	0.09
	GLM+H	0.95	0.97	0.61	0.63	0.61	0.93	0.09

Model (b)

Parameter	Method	$c_{i1,t}$	$c_{i2,t}$	$c_{i3,t}$	$c_{i1,t} \times x_t$	$c_{i2,t} \times x_t$	$c_{i3,t} \times x_t$	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	0.26	0.31	0.29	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	0.26	0.30	0.29	0.72	0.03
	GLM	0.89	0.58	0.66	0.75	0.70	0.83	0.19
	GLM+H	0.89	0.55	0.62	0.76	0.69	0.83	0.19
$P_c = 100$	Lasso	0.33	0.25	0.29	0.33	0.31	0.72	0.02
	Lasso+H	0.30	0.26	0.25	0.28	0.27	0.72	0.02
	ENet	0.33	0.25	0.30	0.33	0.31	0.72	0.02
	ENet+H	0.31	0.26	0.25	0.28	0.27	0.72	0.02
	GLM	0.89	0.53	0.60	0.73	0.65	0.83	0.12
	GLM+H	0.89	0.50	0.57	0.73	0.65	0.83	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 generalized linear model with group lasso (GLM), respectively. "+H" indicates the use of Huber loss instead of the l_2 loss. Column "Noise" reports the average selection frequency of the remaining $P - 6$ 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 $P - 6$ covariates for models (a) and (b) in Table A.2, 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

$(c_{i1,t},\ c_{i2,t},\ c_{i3,t} \times x_t)$ are selected in over 90% of the sample paths, whereas correlated yet redundant covariates $(c_{i3,t},\ c_{i1,t} \times x_t,\ c_{i2,t} \times x_t)$ 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 $P-6$ ones.

Finally, we report the average VIPs of the 6 particular covariates and the average of the remaining $P-6$ covariates for models (a) and (b) in Table A.3, using random forest (RF) and gradient boosted regression trees (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 $P-6$ ones. All methods work equally well.

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.

B Sample Splitting

We consider a number of sample splitting schemes studied in the forecast evaluation literature (see, e.g., West, 2006). The "fixed" scheme 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.

A common alternative to the fixed split scheme 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 training and validation sample fixed. For each rolling window, one refits 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 refitting the entire model once per year, and making out-of-sample predictions using the same fitted model over the subsequent year. Each time we refit, we increase the training sample by a year, while maintaining a fixed size rolling sample for validation. We choose to *not*

Table A.3: Comparison of Average Variable Importance in Simulations

Model (a)

Parameter	Method	$c_{i1,t}$	$c_{i2,t}$	$c_{i3,t}$	$c_{i1,t} \times x_t$	$c_{i2,t} \times x_t$	$c_{i3,t} \times x_t$	Noise
$P_c = 50$	RF	22.97	22.35	4.51	6.24	6.18	22.46	0.16
	GBRT	26.65	24.86	3.79	6.21	6.77	27.70	0.04
	GBRT+H	26.19	25.50	3.84	6.30	6.49	27.74	0.04
	NN1	27.35	27.09	3.80	4.27	3.67	24.80	0.10
	NN2	26.94	26.33	3.81	4.29	3.79	25.73	0.10
	NN3	26.11	25.88	4.08	4.52	3.76	25.43	0.11
	NN4	26.42	25.92	3.87	4.37	3.89	24.55	0.12
	NN5	25.85	25.53	3.89	3.98	3.63	24.78	0.13
$P_c = 100$	RF	22.28	23.21	4.02	5.90	5.71	21.79	0.09
	GBRT	26.16	24.28	4.00	6.25	6.58	26.88	0.03
	GBRT+H	26.30	25.41	3.89	6.23	6.15	27.37	0.02
	NN1	26.00	25.79	3.43	3.77	3.40	23.89	0.07
	NN2	25.56	25.28	3.46	3.76	3.32	24.58	0.07
	NN3	25.53	25.14	3.58	3.99	3.53	23.65	0.08
	NN4	25.23	24.93	3.48	3.91	3.15	24.34	0.08
	NN5	24.84	24.02	3.47	3.84	3.25	23.55	0.09

Model (b)

Parameter	Method	$c_{i1,t}$	$c_{i2,t}$	$c_{i3,t}$	$c_{i1,t} \times x_t$	$c_{i2,t} \times x_t$	$c_{i3,t} \times x_t$	Noise
$P_c = 50$	RF	27.21	6.53	3.54	9.29	5.27	30.74	0.19
	GBRT	33.72	7.89	3.32	10.36	6.59	34.40	0.04
	GBRT+H	34.16	7.89	3.28	10.76	6.95	34.33	0.03
	NN1	56.41	13.97	3.52	3.88	3.10	11.09	0.09
	NN2	51.11	13.41	3.30	3.79	2.71	16.78	0.09
	NN3	51.93	13.42	3.04	3.36	2.78	15.34	0.11
	NN4	50.67	13.08	3.18	3.80	2.94	14.69	0.12
	NN5	47.55	13.00	3.24	4.26	3.02	14.11	0.16
$P_c = 100$	RF	27.13	5.95	3.64	8.68	4.96	30.42	0.10
	GBRT	34.33	7.76	3.53	9.82	6.08	34.61	0.02
	GBRT+H	33.69	7.88	3.38	10.40	6.76	34.16	0.02
	NN1	52.53	12.87	3.12	3.74	2.98	10.56	0.07
	NN2	50.24	12.43	3.04	3.32	2.53	14.23	0.07
	NN3	48.90	12.35	2.86	3.51	2.69	13.04	0.09
	NN4	46.00	11.66	2.92	3.52	2.78	13.32	0.10
	NN5	39.16	10.65	3.57	3.78	2.76	13.53	0.14

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 trees (GBRT), and five architectures of neural networks (NN1,...,NN5), respectively. "+H" indicates the use of Huber loss instead of the l_2 loss. Column "Noise" reports the average variable importance of the remaining $P - 6$ 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.

cross-validate in order to maintain the temporal ordering of the data for prediction.

C Algorithms in Details

C.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

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

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

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

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 (C.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 (C.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 (C.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.

C.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:

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

    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 $\{(z_{i,t}, r_{i,t+1}), (i, t) \in \mathit{Bootstrap}(b)\}$ 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.

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

C.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 (Ioffe and Szegedy (2015)), 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: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i) : i = 1, 2 \ldots, N\}$.

D List of Characteristics

Table A.4: 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.4: 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	lev	Leverage	Bhandari	1988, JF	Compustat	Annual
44	lgr	Growth in long-term debt	Richardson, Sloan, Soliman & Tuna	2005, JAE	Compustat	Annual
45	maxret	Maximum daily return	Bali, Cakici & Whitelaw	2011, JFE	CRSP	Monthly
46	mom12m	12-month momentum	Jegadeesh	1990, JF	CRSP	Monthly
47	mom1m	1-month momentum	Jegadeesh & Titman	1993, JF	CRSP	Monthly
48	mom36m	36-month momentum	Jegadeesh & Titman	1993, JF	CRSP	Monthly
49	mom6m	6-month momentum	Jegadeesh & Titman	1993, JF	CRSP	Monthly
50	ms	Financial statement score	Mohanram	2005, RAS	Compustat	Quarterly
51	mvel1	Size	Banz	1981, JFE	CRSP	Monthly
52	mve_ia	Industry-adjusted size	Asness, Porter & Stevens	2000, WP	Compustat	Annual
53	nincr	Number of earnings increases	Barth, Elliott & Finn	1999, JAR	Compustat	Quarterly
54	operprof	Operating profitability	Fama & French	2015, JFE	Compustat	Annual
55	orgcap	Organizational capital	Eisfeldt & Papanikolaou	2013, JF	Compustat	Annual
56	pchcapx_ia	Industry adjusted % change in capital expenditures	Abarbanell & Bushee	1998, TAR	Compustat	Annual
57	pchcurrat	% change in current ratio	Ou & Penman	1989, JAE	Compustat	Annual
58	pchdepr	% change in depreciation	Holthausen & Larcker	1992, JAE	Compustat	Annual
59	pchgm_pchsale	% change in gross margin - % change in sales	Abarbanell & Bushee	1998, TAR	Compustat	Annual
60	pchquick	% change in quick ratio	Ou & Penman	1989, JAE	Compustat	Annual
61	pchsale_pchinvt	% change in sales - % change in inventory	Abarbanell & Bushee	1998, TAR	Compustat	Annual
62	pchsale_pchrect	% change in sales - % change in A/R	Abarbanell & Bushee	1998, TAR	Compustat	Annual

Table A.4: Details of the Characteristics (Continued)

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