Evaluation of Machine Learning in Empirical Asset Pricing

Ze Yu Zhong

Sunday 12th May, 2019

Contents

1. Introduction

1.1. Topic

This thesis aims to evaluate the application of machine learning algorithms in empirical asset pricing. While there has been a significant recent interest in applying machine learning to the problem of predicting asset returns, there is little literature that focuses on how well these algorithms are at capturing true underlying variables in determining stock returns. 12 different simulated datasets ranging from linear to highly non-linear data generating processes incorporating observed phenomena of cross sectional correlation, persistence, and stochastic volatility, in addition to real world data will be used to assess the performance of linear models, elastic net models, random forests and neural networks. Model performance will be assessed according to their out of sample predictive R^2 and errors, in addition to whether or not they were able to identify the correct variables in the data generating process according to a variable importance metric.

1.2. Background Literature and Motivations

This paper is motivated by evaluating the performance of machine learning algorithms in empirical asset pricing, focusing on how well they deal with the many unique problems in financial returns data. Here, we define "performance" to refer to two forms of metrics conventional in the literature: R^2 (and more specifically, out of sample R^2), and forecast errors (see 2.10 for more details).

We first begin by defining "factors" with the more contemporary definition as suggested by Harvey et al. (2016): a collection of regressors to be used in pricing returns that can be used to proxy for unknown underlying risk factors due to their correlation with cross sectional returns. Most notably, their definition rejects the more strict view that risk factors should be variables that have unpredictable variation through time, and that they should be able explain cross sectional return patterns. Harvey et al. (2016) further groups factors into the two broad categories of "common" and individual firm "characteristics." "Common" factors under this definition can be viewed as proxies for sources of risk constant across all firms, such as macroeconomic variables. They note that individual firm characteristics are unlikely to satisfy the more strict definition because they are often pre-known and display limited time series variation.

Because of this less strict definition, factors introduced and used in the literature often exhibit properties which makes them unsuitable for inclusion in models, such as high persistence, high levels of non-stationarity and cross sectional correlation.

Goetzmann and Jorion (1993) and Ang and Bekaert (2006) note the persistence present in dividend ratio factors. This means that movements in dividend ratios are dominated by movements in price and therefore dividend ratios are correlated with lagged dependent variables on the right hand side of the

regression equation. This violates the assumptions of independent regressors required for traditional regression models (ordinary least squares) to be unbiased, resulting in t statistics which are biased upwards and increase with time horizon due to autocorrelated errors. Importantly, Goetzmann and Jorion (1993) show that corrections to t statistics using the Generalized Method of Moments and Newey-West standard errors also appear to be biased upwards, making them unreliable.

Goyal and Welch (2003) provide a more comprehensive study on the performance of lagged dividend price ratios, with specific focus on out of sample predictive performance both in terms of R^2 and forecast errors. They conclude that while models incorporating dividend related factors were able to achieve higher in sample performance prior to 1990 than the historical mean, they could not have outperformed the historical mean *out of sample*. Goyal and Welch (2003) attribute this to the increasing persistence and non-stationarity of dividend ratios, noting that they have become like random walks as of 2001. This mirrors the sentiment of (Lettau and Ludvigson (2001), Schwert (2003) and others) who conclude that models incorporating dividend ratios seemed to break down in the 2000s due to a changing economic environment despite having performed well in the 1990s.

Despite the controversy, the prevailing tone within the literature was that various factors such as dividend ratios, earnings price ratio, interest and inflation and other financial indicators were able to predict excess returns, with Lettau and Ludvigson (2001) remarking that this was now "widely accepted." However, Welch and Goyal (2008) extend upon the work of Goyal and Welch (2003) by including a more comprehensive set of variables and time horizons. They conclude that not a single variable had any statistical forecasting power. Crucially, they demonstrate the non-robustness of models incorporating these factors by showing that the significance values of some factors change with the choice of sample periods.

Despite this, the literature has continued to produce more factors: quantitative trading firms were using 81 factor models as the norm by 2014 (Hsu and Kalesnik, 2014), and Harvey and Liu (2019) currently document well over 600 different factors suggested in the literature.

The dramatic increase in the number of factors in of itself poses challenges to traditional statistical techniques. Harvey et al. (2016) detail the false discovery problem when the number of potential factors is extremely high. The significance of a factor in a traditional regression setting is determined by a single hypothesis test, which by construction carries a level of significance α controlling the type I error rate: the probability of finding a factor that is significant but is not. When the number of potential factors is very high, it is very likely that a factor will be concluded as significant by pure chance. Harvey et al. (2016) produce a multiple testing framework to mitigate this, and conclude that many of the historically discovered would have been deemed significant by chance.

Furthermore, Feng et al. (2019) note that the number of potential factors discovered in the literature has increased to the same scale as, if not greater, than the number of stocks considered in a typical portfolio, or the time horizon, producing highly inefficient covariances in a standard cross sectional regression. Moreover, when the number of factors exceeds the sample size, traditional cross sectional regressions become infeasible and do not produce solutions altogether.

It does not help that many factors are cross sectionally correlated, meaning that factors which are discovered to be significant may simply be so because they are correlated with a true, underlying factor and do not provide independent information themselves, a concern which Cochrane (2011) calls the multidimensional challenge. Freyberger et al. (2017) notes that this is especially challenging for traditional regression models, which make strong functional form assumptions and are sensitive to outliers.

More recently, machine learning algorithms have emerged within the literature and appear to be well suited to the task of predicting asset returns. The definition of machine learning can be vague and is

often context specific; Hastie et al. (2009) in *An Introduction to Statistical Learning* describes statistical (machine) learning as a vast set of tools for understanding data, and *supervised* learning specifically as the process of building a statistical model for the prediction or estimation of an output based on input(s). In the context asset pricing, we use the term to refer to a diverse collection of:

1. high-dimensional models for statistical prediction,
2. the "regularization" methods for model selection and mitigation of overfitting input data,
3. and the efficient systematic methods for searching potential model specifications.

The high dimensional and hence flexible nature of machine learning brings more hope to approximating unknown and likely complex data generating processes that underlie excess returns. The flexibility however, comes at a cost of potentially overfitting in sample data (referred to as training data in the machine learning literature), generalizing poorly and producing poor forecasts. The regularization aspect of machine learning explicitly guards against problem and emphasizes out of sample performance. Finally, machine learning offers tools which are designed to produce an an optimal model specification from all possible models with manageable computational cost, all in a systematically consistent way.

Machine learning has seen some applications in the empirical asset pricing literature. Kozak et al. (2017), Rapach and Zhou (2013) and Freyberger et al. (2017) all apply shrinkage and selection methods from machine learning in factor selection.

Most importantly, portfolios constructed using machine learning have been demonstrated to outperform traditional models in predicting stock returns (Gu et al. (2018), Hsu and Kalesnik (2014) and Feng et al. (2018)) in terms of out of sample predictive R^2 and Sharpe Ratios. Gu et al. (2018) attribute this to machine learning's ability to evaluate and consider non-linear complexities among factors that cannot be feasibly achieved using traditional techniques.

However, there is little work done on how machine learning actually recognises and deals with the challenges of returns prediction documented in the literature. Prior work has been done by Gu et al. (2018), however; only basic simulation designs which were not representative of real financial data were considered. In particular, their design did not consider cross sectionally correlated factors.

Furthermore, Feng et al. (2018) in particular use cross validation as part of their model building procedure, destroying the temporal aspect of returns data, in addition to only using a handful of factors. Gu et al. (2018) produce models using a training sample which ends in the 1970s to ultimately produce forecasts for the most recent 30 years. Given the non-robustness of financial data affecting even traditional regressions which are considered to be more inflexible, more research should be done into the robustness of more flexible machine learning methods with regards to sample selection.

For the aspect of factor selection specifically, Gu et al. (2018) concludes that all of the machine methods agree on the same subset of important factors. However, while their factor importance metrics for regression-tree based methods and neural networks (the most complex methods considered) are mostly consistent, they have differences in terms of the relative importance of each factor, in addition to completely different conclusions for the dividend yield factor.

This paper will be the first in focusing on how machine learning algorithms perform in environments with problems exhibited by financial returns data through extending the simulation designs of Gu et al. (2018). In addition, these algorithms will once again be evaluated on real world data, but with only more recent and representative data included in order to test their short term robustness in predicting stock returns. These two aspects of the study together are able to offer a better glimpse as to how "black box" machine learning algorithms deal with the challenges present in asset pricing, if at all.

2. Methodology

2.1. Overall Model Design

Each model will be presented and explained so that a reader without any machine learning background can understand the basic idea behind each model. Details such as the computational and specific algorithm used for each model are included in the Appendix (5.1). This is because there are many variations of algorithms available, and more importantly, specific understanding of how the algorithm works is not necessary.

An an extension to Gu et al. (2018)'s work, we strive to use a similar design. All asset excess monthly returns denoted as $r_{i,t+1}$ are modelled as an additive prediction error model conditional on the true and unobservable information set available to market participants up to and including time t, $\mathcal{F}_t$):

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

where

$$E(r_{i,t+1}|\mathcal{F}_t) = g^*(z_{i,t}) \tag{2}$$

Stocks are indexed as $i = 1, \ldots, N$ and months by $t = 1, \ldots, T$. $g^*(z_{i,t})$ represents the model approximation using the P dimensional predictor set $z_{i,t}$. We allow $g^*(z_{i,t})$ to be a flexible function of the predictor set $z_{i,t}$, and most notably, not depend on i or t directly. This means that we do not re-estimate a model for each time period, or independently estimate a model for each stock. Note that $g^*(z_{i,t})$ only contains information in time t for individual stock i, meaning that while the model and its parameters will be estimated using $\mathcal{F}_t$ for stock i, predictions for $r_{i,t+1}$ will only use information at time t as an input, analogous to using variables lagged by one period.

2.2. Sample Splitting

Imperative to any machine learning technique is the establishment of how the dataset is to be split into training, validation and test sets. The training set is used to initially build the model and provide initial estimates of parameters, whereas the validation set is used to tune model parameters to optimise out of sample performance, thus preventing overfitting. The validation set acts as a simulation of out of sample testing, whereas the test set is used only for evaluation, and is thus truly out of sample.

There are three main approaches to splitting temporal data (such as financial data).

The first is to decide arbitrarily on a single training, validation and test set. This method is straightforward and the least computationally intensive, but is limited and inflexible in evaluating how models perform when more recent data is provided for training.

The second method is a "rolling window" method, where a fixed size or "window" for the training and validation set is first chosen. This window then incrementally move forwards in time to include more recent data, with a set of forecasts for the test sets made for all possible windows.

The third is a "recursive" method, which is the same as the rolling window method, but different in that the training set always contains previous data, with only the validation set staying fixed in size and "rolling" forwards. Hence, it is also referred to as a "growing window."

Both the rolling window and recursive schemes are very computationally intensive. Therefore, a hybrid of the rolling and recursive schemes was considered: the training set is increased by one year with each refit, the validation set remains one year in length but moves forward by one year, and forecasts are

made using that model for the subsequent year. Cross validation was not done to maintain the temporal ordering of the data.

2.3. Loss Function

The choice of the loss function used in models is imperative to machine learning. The loss functions considered are Mean Absolute Error (MAE), Mean Squared Error and Root Mean Squared Error (MSE, and RMSE) and Huber Loss.

2.3.1. Mean Absolute Error

The mean absolute error (MAE) is simply the average magnitude of errors. Because of this, it places equal weighting to all magnitudes of errors and is more robust to outliers.

$$\text{MAE} = \frac{1}{n}\sum_{j=i}^{n}|y_j - \hat{y_j}| \tag{3}$$

2.3.2. Mean Squared Error and Root Mean Squared Error

The mean squared error (MSE) and root mean squared error (RMSE) are quadratic scoring methods. This means that they place higher weight on large errors. Models that minimize this metric are therefore more sensitive to outliers.

$$\text{MSE} = \frac{1}{n}\sum_{j=i}^{n}(y_j - \hat{y_j})^2 \tag{4}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=i}^{n}(y_j - \hat{y_j})^2} \tag{5}$$

2.4. Linear Model

The least complex model considered is the simple linear regression model, also known that ordinary least squares in the case of the using mean squared error as the loss function. Linear models struggle with high-dimensionality. Nevertheless, despite being expected to perform poorly it was implemented as a "control."

The simple linear model assumes that the underlying conditional expectation $g^*(z_{i,t})$ can be modelled as a linear function of the predictors and the parameter vector θ:

$$g(z_{i,t};\theta) = z'_{i,t}\theta \tag{6}$$

This model can capture non-linearities only if the predictor set $z^*_{i,t}$ contains specified non-linear transformations or interaction terms.

Note that computing this model with respect to minimizing the mean squared error yields the pooled ordinary least squares estimator (POLS).

2.5. Penalized Linear Model

Penalized linear models have the same underlying statistical model as simple linear models, but differ in their addition of a new penalty term in the loss function:

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

Several choices exist for the choice of the penalty function $\phi(\theta;.)$. This focus of this paper is the popular "elastic net" penalty (Zou and Hastie, 2005):

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

The elastic net has two hyperparameters: λ, which controls the overall magnitude of the loss, and ρ, which controls the shape of the penalization.

The $\rho = 1$ case corresponds to ridge regression proposed by Hoerl and Kennard (1970), which uses l_2 that shrinks all coefficients closer to 0, but not to 0. Ridge regression is therefore a shrinkage method which prevents coefficients from becoming too large and overpowering. For $0 < \rho < 1$, the elastic net aims to produce parsimonious models through both shrinkage and selection.

The $\rho = 0$ case corresponds to the popular LASSO and uses absolute (l_1) parameter penalization proposed by Tibshirani (1996), which geometrically allows the coefficients to be shrunk to 0. This allows it to impose sparsity, and can be thought of as a variable selection tool.

By combining the properties of LASSO and ridge regression, the elastic net aims to produce parsimonious models through both coefficient shrinkage and selection.

The hyperparameters λ and ρ are both tuned using the validation sample (see 5.1).

2.6. Classification and Regression Trees

Classification and regression trees are fully non-parametric models that can capture complex multi-way interactions. A tree "grows" in a series of iterations. With each iteration, a split ("branch") is made along one predictor such that it is the best split available at that stage with respect to minimizing the loss function. These steps are continued until each observation is its own node, or more commonly until the stopping criterion is met. The eventual model slices the predictor space into rectangular partitions, and predicts the unknown function $g^*(z_{i,t})$ with the average value of the outcome variable in each partition.

The prediction of a tree, $\mathcal{T}$, with K "leaves" (terminal nodes), and depth L is

$$g(z_{i,t};\theta,K,L) = \sum_{k=1}^{K} \theta_k \mathbf{1}_{z_{i,t}\in C_k(L)} \tag{9}$$

where $C_k(L)$ is one of the K partitions in the model.

For this study, only recursive binary trees (the most common and easy to implement) are considered. Though trees were originally proposed and fit with respect to minimizing mean squared error, they can be grown with respect to a variety of loss functions, including mean absolute error, mean squared error:

$$H(\theta,C) = \frac{1}{|C|}\sum_{z_{i,t}\in C} L(r_{i,t+1}-\theta) \tag{10}$$

where $|C|$ denotes the number of observations in set C (partition). Given C, it is clear that the optimal choice for minimising the loss function when it is mean squared error is simply $\theta = \frac{1}{|C|}\sum_{z_{io,t}\in C} r_{i,t+1}$ i.e. the average of the partition, and the median of the partition when the loss function is mean absolute error.

Trees, grown to a deep enough level, are highly unbiased and flexible. The tradeoff of course, is their high variance and instability. Thus, an ensemble method called "Random Forest" was proposed by Breiman (2001) to regularize trees by combining many different trees into a single prediction.

2.7. Random Forests

Random Forests are an extension of trees that attempt to address some of their problems. A random forest algorithm creates B different bootstrap samples from the training dataset, fits an overfit (and hence low bias) regression tree to each using only a random subset m size from all available predictors (also known as dropout), and then averages their forecasts as the final output. The overfit trees means that the underlying trees has low bias, and the dropout procedure means that they have low correlation. Thus, averaging these low bias, uncorrelated trees results in a low bias, yet stable model. Specific details of the random forest algorithm are detailed in the appendix.

2.8. Neural Networks

2.8.1. Introduction

Neural networks have theoretical underpinnings as "universal approximators" for any smooth predictive association, (Hornik et al. (1989)). They are arguably the most complex type of model available, able to capture several non-linear interactions through their many layers, hence its other name "deep learning." On the flipside, their high flexibility often means that they are among the most parameterized and least interpretable models, earning them the reputation as a black box model.

The scope of this paper is limited to traditional "feed-forward" networks. The feed forward network consists of an "input layer" of scaled data inputs, one or more "hidden layers" which interact and non-linearly transform the inputs, and finally an output layer that aggregates the hidden layers and transform them a final time for the final output.

More specifcally, a neural network consists of layers denoted by $l = 0, 1, \ldots, L$, with $l = 0$ denoting the input layer and $l = L$ denoting the output layer, and $K^{(l)}$ denoting the number of neurons in each hidden layer. The input layer is defined using predictors, $x^{(0)} = (1, z_1, \ldots, z_N)'$. The output of neuron k in layer l is then $x_k^{(l)}$. Next, define the vector of outputs for this layer as $x^{(l)} = (1, x_1^{(l)}, \ldots, x_{K^{(l)}}^{(l)})'$. The recursive output formula for the neural network at each neuron in layer $l > 0$ is then:

$$x_k^{(l)} = \alpha(x^{(l-1)'}\theta_k^{l-1}), \tag{11}$$

where $\alpha()$ represents the activation function for that layer (see next section) with the final output

$$g(z;\theta) = x^{(L-1)'}\theta^{L-1} \tag{12}$$

Note that the specification of a constant "1" at the beginning of each layer is the same as specifying a bias term as is popular in other parametrizations.

Neural networks with up to 5 hidden layers were considered, each named NNX where X represents the number of hidden layers. The number of neurons is each layer was chosen according to the geometric pyramid rule (Masters, 1993): NN1 has 32 neurons, NN2 has 32 and 16 neurons in the first and second hidden layers respectively, NN3 has 32, 16, and 8 neurons, NN4 has 32, 16, 8, and 4 neurons, and NN5 has 32, 16, 8, 4, 2 neurons respectively. All units are fully connected; that is, each neurons receives input from all neurons the layer before it (see Figure 1). This mimics the methodology in Gu et al. (2018).

2.8.2. Activation Function

The ReLU activation function

$$\text{ReLU}(x) = max(0, x) \tag{13}$$

was used for all hidden layers owing to its high computational speed, and hence popularity within recent literature (see Lecun et al. (2015) and Ramachandran et al. (2017), among others). Other potential choices for activation functions such as sigmoid, softmax, tanh etc. were not used due to the additional complexities and computational costs involved with validating the choice of activation function.

2.8.3. Computation

The neural network's weight and bias parameters for each layer are estimated by minimizing the loss function with respect to the parameters, i.e. by calculating the partial derivative with respect to a

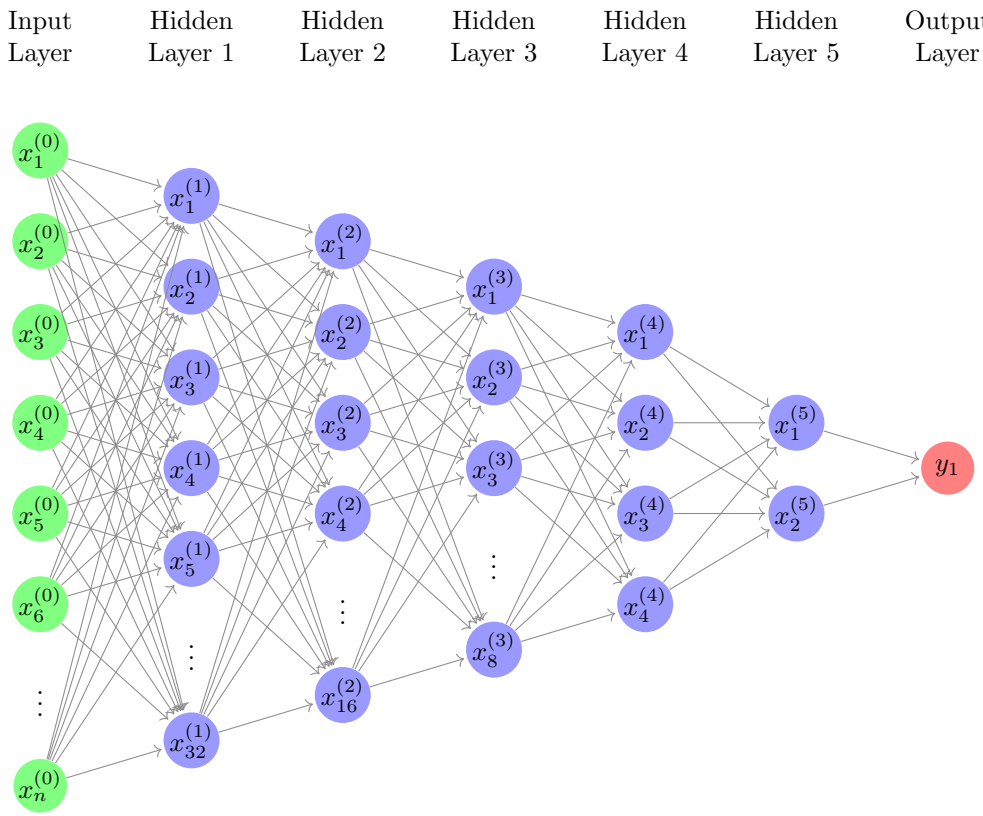

Figure 1. Neural Network 5 (most complex considered), without biases terms drawn

specific weight or bias element.

The solution to this is typically found via backpropagation, an iterative alogrithm similar to Gauss-Newton steps and produces a local minimum. These steps can be visualized as a descent towards the local minimum of the loss function; it is thus also known as "gradient descent." Note that the lack of a global minimum is actually desirable, as global minimums tend to be overfit solutions to the problem, (Choromanska et al., 2014). This is to be calculated and averaged across all training observations, and is thus extremely computationally intensive.

A common solution is to use "stochastic gradient descent" (SGD) where instead of optimising the loss function with respect to the entire training sample, only a small, random subset of the data (batch) is used at each optimisation step. This sacrifices some accuracy for a dramatic improvement in computational speed.

Due the noisiness (randomness) introduced by SGD, the path towards the local minimum is more of a quick "zig zag" and has the potential to "overshoot" the local minimum and not converge to a solution. This is typically controlled via a hyperparameter known as the learning rate, which controls the step size of each gradient descent iteration. Note that it is also common to apply a learning rate hyperparameter in non SGD environments as a way to control the magnitude of each step towards the solution to assist with convergence. The learning rate is to be tuned so that the solution path descends quickly enough to be computationally feasible, but slow enough so that it does not overshoot the local minimum and not converge. In this paper, a learning rate shrinkage algorithm which adaptively shrinks the learning rate as convergence occurs was employed (see Kingma and Ba (2014)).

2.8.4. Batch Normalization

"Batch normalization" is a technique for addressing a phenomenon known as internal covariate shift, a particularly prevalent problem in training deep, complex neural networks, (Ioffe and Szegedy, 2015). Internal covariate shift occurs when the distributions of each layers' inputs change as the parameters of the previous layer change, resulting in the need for much slower learning rates and more careful initialization of parameters. By normalizing (de-meaning and variance standardizing) each training step (batch) input, the representation power of each neuron (unit) is restored. Additionally, significant gains in computational speed may also be achieved.

2.8.5. Initialization

Finally, multiple random starting values for the weights and biases (seeds) were used in training neural networks, with the resulting predictions averaged in an ensemble model, Hansen and Salamon (1990). This regularizes the variance associated with the initial starting values for the weights and biases. It should be noted however, that contemporary literature suggests that all local solutions are of similar quality and thus this is not strictly necessary, (Choromanska et al., 2014).

2.9. Simulation Design

2.9.1. Overall Design

RS: double check conformability of everything, make sure that the errors are corrected

Though Gu et al. (2018) explore the performance of machine learning on simulated returns series, their design used factors are uncorrelated across i, and, in particular, that the factors which enter the return equation are uncorrelated with the factors that do not enter the return equation. As note by Harvey et al. (2016) and many others, this is not what is observed in practice.

Therefore, we simulate an extension: a latent factor model with stochastic volatility for excess return, r_{t+1}, for $t = 1, \ldots, T$:

$$r_{i,t+1} = g\left(z_{i,t}\right) + \beta_{i,t+1} v_{t+1} + e_{i,t+1}; \quad z_{i,t} = (1, x_t)' \otimes c_{i,t}, \quad \beta_{i,t} = (c_{i1,t}, c_{i2,t}, c_{i3,t}) \tag{14}$$

$$e_{i,t+1} = \exp\left(\sigma^2_{i,t+1}\right) \varepsilon_{i,t+1}; \tag{15}$$

$$\sigma^2_{i,t+1} = \omega + \gamma \sigma^2_t + w_{t+1} \tag{16}$$

Let v_{t+1} be a 3×1 vector of errors, and $w_{t+1} \sim N(0,1)$ and $\varepsilon_{i,t+1} \sim N(0,1)$ scalar error terms. The parameters of these are tuned such that the R squared for each individual return series was 50% and annualized volatility 30%.

The matrix C_t is an $N \times P_c$ vector of latent factors, where the first three columns correspond to $\beta_{i,t}$, across the $1 \leq i \leq N$ dimensions, while the remaining $P_c - 3$ factors do not enter the return equation. The $P_x \times 1$ vector x_t is a 3×1 multivariate time series, and ε_{t+1} is a $N \times 1$ vector of idiosyncratic errors.

2.9.2. Simulating Characteristics

A simulation mechanism for C_t that gives some correlation across the factors and across time was used. First consider drawing normal random numbers for each $1 \leq i \leq N$ and $1 \leq j \leq P_c$, according to

$$\overline{c}_{ij,t} = \rho_j \overline{c}_{ij,t-1} + \epsilon_{ij,t}; \quad \rho_j \sim \mathcal{U}\left(\frac{1}{2}, 1\right) \tag{17}$$

Then, define the matrix

$$B := \Lambda\Lambda' + \frac{1}{10}\mathbb{I}_n, \quad \Lambda_i = (\lambda_{i1}, \ldots, \lambda_{i4}), \quad \lambda_{ik} \sim N(0,1),\ k = 1, \ldots, 4 \tag{18}$$

which we transform into a correlation matrix W via

$$W = (\text{diag}(B))^{\frac{-1}{2}} (B) (\text{diag}(B))^{\frac{-1}{2}} \tag{19}$$

To build in cross-sectional correlation, from the $N \times P_c$ matrix $\bar{C}_t$, we simulate characteristics according to

$$\widehat{C}_t = W\overline{C}_t \tag{20}$$

Finally, the "observed" characteristics for each $1 \leq i \leq N$ and for $j = 1, \ldots, P_c$ are constructed according to:

$$c_{ij,t} = \frac{2}{n+1} \text{rank}\left(\hat{c}_{ij,t}\right) - 1. \tag{21}$$

with the rank transformation normalizing all predictors to be within $[-1, 1]$.

2.9.3. Simulating Macroeconomic Series

For simulation of x_t, a 3×1 multivariate time series, we consider a Vector Autoregression (VAR) model, a generalization of the univariate autoregressive model to multiple time series:

$$x_t = Ax_{t-1} + u_t, \quad u_t \sim N\left(\mu = (0,0,0)', \Sigma = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\right)$$

where we have three separate specifications for the matrix A:

$$(1)\ A = \begin{pmatrix} .95 & 0 & 0 \\ 0 & .95 & 0 \\ 0 & 0 & .95 \end{pmatrix} \quad (2)\ A = \begin{pmatrix} 1 & 0 & .25 \\ 0 & .95 & 0 \\ .25 & 0 & .95 \end{pmatrix} \quad (3)\ A = \begin{pmatrix} .99 & .2 & .1 \\ .2 & .90 & -.3 \\ .1 & -.3 & -.99 \end{pmatrix} \tag{22}$$

2.9.4. Simulating Return Series

We will consider four different functions for $g(z_{i,t})$:

$$(1)\ g_1\left(z_{i,t}\right) = \left(c_{i1,t}, c_{i2,t}, c_{i3,t} \times x_t'\right)\theta_0; \quad \theta_0 = (0.02, 0.02, 0.02)'$$
$$(2)\ g_2\left(z_{i,t}\right) = \left(c_{i1,t}^2, c_{i1,t} \times c_{i2,t}, \text{sgn}\left(c_{i3,t} \times x_t'\right)\right)\theta_0; \quad \theta_0 = (0.04, 0.035, 0.01)'$$
$$(3)\ g_3\left(z_{i,t}\right) = \left(1[c_{i3,t} > 0], c_{i2,t}^3, c_{i1,t} \times c_{i2,t} \times 1[c_{i3,t} > 0], \text{logit}\left(c_{i3,t}\right)\right)\theta_0; \quad \theta_0 = (0.04, 0.035, 0.01, 0.01)'$$
$$(4)\ g_4\left(z_{i,t}\right) = \left(\hat{c}_{i1,t}, \hat{c}_{i2,t}, \hat{c}_{i3,t} \times x_t'\right)\theta_0; \quad \theta_0 = (0.02, 0.02, 0.02)'$$

$g_1\left(z_{i,t}\right)$ allows the characteristics to enter the return equation linearly, and $g_2\left(z_{i,t}\right)$ allows the characteristics to enter the return equation interactively and non-linearly. These two specifications correspond

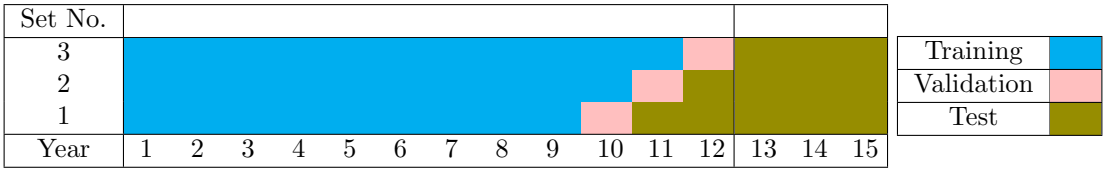

Figure 2. Sample Splitting Procedure

to the simulation design proposed by Gu et al. (2018).

$g_3(z_{i,t})$ allows the characteristics to enter in a highly complex and non-linear fashion.

$g_4(z_{i,t})$ builds returns using $\hat{c}$, which are the original characteristics without cross sectional correlation built in.

θ^0 will be tuned such that the cross sectional R^2 for any given time period is 25%, and the predictive R^2 across the entire time period is 5%.

The simulation design results in $3 \times 4 = 12$ different simulated datasets, each with $N = 200$ stocks, $T = 180$ periods and $P_c = 100$ characteristics. Each design was simulated 50 times to assess the robustness of machine learning algorithms.

2.9.5. Sample Splitting

$T = 180$ monthly periods corresponds to 15 years. The training sample was set to start from $T = 108$ or 9 years, a validation set 1 year in length. The last 3 years were reserved as a test set never to be used for validation or training. We employ the hybrid growing window approach as described earlier in section 2.2 (see Figure 2 for a graphical representation).

Unlike Gu et al. (2018) who split their simulated dataset into training, validation and test sets of equal length, we choose a splitting scheme that is consistent with the splitting scheme to be used for the real world dataset.

2.10. Model Evaluation

2.10.1. R Squared

Overall predictive performance for individual excess stock returns were assessed using the out of sample R^2:

$$R^2_{OOS} = 1 - \frac{\sum_{(i,t)\in\mathcal{T}_3}(r_{i,t+1} - \widehat{r}_{i,t+1})^2}{\sum_{(i,t)\in\mathcal{T}_3}(r_{i,t+1} - \bar{r}_{i,t+1})^2} \tag{23}$$

where $\mathcal{T}_3$ indicates that the fits are only assessed on the test subsample, which is never used for training or tuning.

2.10.2. Diebold-Mariano Test

The Diebold-Mariano test (Diebold and Mariano (2002), and Harvey et al. (1997)) is a procedure which compares the forecast accuracy of two forecast methods. It is different to the overall R squared metric because it tests whether or not the models' forecast accuracy is significantly different.

Under the null hypothesis of the Diebold-Mariano test:

$$S_1^* = \left[\frac{n+1-2h+n^{-1}h(h-1)}{n}\right]^{1/2} S_1; \quad S_1^* \sim N(0,1) \tag{24}$$

$$S_1 = \left[\hat{V}(\bar{d})\right]^{-1/2} \bar{d} \tag{25}$$

$$\hat{\gamma}_k = n^{-1} \sum_{t=k+1}^{n} (d_t - \bar{d})(d_{t-k} - \bar{d}) \tag{26}$$

$$V(\bar{d}) \approx n^{-1} \left[\gamma_0 + 2\sum_{k=1}^{h-1} \gamma_k\right] \tag{27}$$

where d_t represents the difference series between the forecast errors of the two models $e_1t - e_2t$, $\hat{\gamma}_k$ represents the sample kth autocovariance for d_t, and S_1 represents the original unmodified Diebold Mariano test statistic.

As all models in this paper will be producing forecasts for an entire cross section of stocks, e_1t and e_2t will instead represent the average forecast errors for each model.

2.11. Variable Importance

The importance of each predictor j is denoted as VI_j, and is defined as the reduction in predictive R-Squared from setting all values of predictor j to 0, while holding the remaining model estimates fixed. This will allow us to see which variables each model determines to be important.

3. Study

3.1. Data

The dataset will be provided by the CRSP/Compustat database and include only more recent returns data, specifically before and after the global financial crisis. All stocks were included in order to alleviate any biases associated with small firms or survivorship.

3.2. Model

All machine learning methods are designed to approximate the empirical model $E_t(r_{i,t+1}) = g * (z_{i,t})$ defined in equation (2). The baseline set of stock-level covariates $z_{i,t}$ as:

$$z_{i,t} = x_t \otimes c_{i,t} \tag{28}$$

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 (and are this common to all stocks, including a constant). $z_{i,t}$ is a $P \times 1$ vector of features for predicting individual stock returns ($P = P_c P_x$) and includes interactions between individual characteristics and macroeconomic characteristics.

4. Results

All work has been and will be done in R.

Work is currently being done on tuning the simulated datasets.

If there is enough time and computing resources, Long Term Short Memory models, an extension of neural networks which includes the output of a neural network trained on previous data, will also be evaluated.

5. Appendix

5.1. Algorithms

5.1.1. Penalized Linear

5.1.2. Classification and Regression Trees

For full details of the Classification and Regression Tree algorithm see Breiman (1984).

Algorithm 1: Classification and Regression Tree

Initialize ;

for *d from 1 to L* **do**

  for *i in* $C_l(d-1), l = 1, \ldots, 2^{d-1}$ **do**

    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}:

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

    Define the impurity function:

$$\mathcal{L}(C, C_{left}, C_{right}) = \frac{|C_{left}|}{|C|} H(C_{left}) + \frac{|C_{right}|}{|C|} H(C_{right})$$

    where

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

    and $|C|$ denotes the number of observations in set C

    Find the optimal split

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

    Update nodes (partition the data):

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

  end

end

Result: The prediction of a regression tree is:

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

5.1.3. *Random Forest*

For full details of the Random Forest algorithm see Breiman (2001).

Algorithm 2: Random Forest

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

Draw bootstrap samples $(z_{i,t}, r_{i,t+1}), (i,t) \in Bootstrap(b)$ from the dataset Grow a tree T_b using Algorithm, using only a random subsample, say $\sqrt{P}$ of all features Denote the resulting bth tree as

$$\hat{g}_b(z_{i,t}, \hat{\theta}_b, L) = \sum_{k=1}^{2^L} \theta_b^k \mathbf{1}_{z_{i,t} \in C_k(L)}$$

end

Result: The final random forest prediction is given by the output of all trees:

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

5.1.4. *Neural Networks*

ADAM algorithm for stochastic gradient descent and learning rate shrinkage as detailed by Kingma and Ba (2014).

Algorithm 3: Early stopping via validation

Initialize $j = 0$, $\epsilon = \infty$ and select the patience parameter p (max iterations)
while j ¡ p **do**
  Update θ using the training algorithm 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: θ' is the final parameter estimate

Batch Normalization Algorithm as detailed by Ioffe and Szegedy (2015).

Algorithm 4: Batch Normalization for one activation over one batch

Input: Values of x for each activation over a batch $\mathcal{B} = x_1, x_2, \ldots, x_N$
$\mu_{\mathcal{B}} \leftarrow \frac{1}{N} \sum_{i=1}^{N} x_i$
$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu_{\mathcal{B}})^2$
$\hat{x_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$
$y_i \leftarrow \gamma \hat{x_i} + \beta := BN_{\gamma,\beta}(x_i)$
Result: $y_i = BN_{\gamma,\beta}(x_i) : i = 1, 2, ..., N$

References

Ang, A., Bekaert, G., 2006. Stock return predictability: Is it there? The Review of Financial Studies 20, 651–707.

Breiman, L., 1984. Classification and Regression Trees. Routledge.

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

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., LeCun, Y., 2014. The Loss Surfaces of Multilayer Networks. arXiv:1412.0233 [cs] ArXiv: 1412.0233.

Cochrane, J. H., 2011. Presidential Address: Discount Rates. The Journal of Finance 66, 1047–1108.

Diebold, F. X., Mariano, R. S., 2002. Comparing predictive accuracy. Journal of Business & economic statistics 20, 134–144.

Feng, G., Giglio, S., Xiu, D., 2019. Taming the factor zoo: A test of new factors. Tech. rep., National Bureau of Economic Research.

Feng, G., He, J., Polson, N. G., 2018. Deep Learning for Predicting Asset Returns. arXiv:1804.09314 [cs, econ, stat] ArXiv: 1804.09314.

Freyberger, J., Neuhierl, A., Weber, M., 2017. Dissecting characteristics nonparametrically. Tech. rep., National Bureau of Economic Research.

Goetzmann, W. N., Jorion, P., 1993. Testing the predictive power of dividend yields. The Journal of Finance 48, 663–679.

Goyal, A., Welch, I., 2003. Predicting the equity premium with dividend ratios. Management Science 49, 639–654.

Gu, S., Kelly, B., Xiu, D., 2018. Empirical asset pricing via machine learning. Tech. rep., National Bureau of Economic Research.

Hansen, L. K., Salamon, P., 1990. Neural network ensembles. IEEE Transactions on Pattern Analysis & Machine Intelligence pp. 993–1001.

Harvey, C. R., Liu, Y., 2019. A Census of the Factor Zoo. Social Science Research Network p. 7.

Harvey, C. R., Liu, Y., Zhu, H., 2016. ... and the Cross-Section of Expected Returns. The Review of Financial Studies 29, 5–68.

Harvey, D., Leybourne, S., Newbold, P., 1997. Testing the equality of prediction mean squared errors. International Journal of Forecasting 13, 281–291.

Hastie, T., Tibshirani, R., Friedman, J. H., 2009. The elements of statistical learning: data mining, inference, and prediction. Springer series in statistics, Springer, New York, NY, second ed.

Hoerl, A. E., Kennard, R. W., 1970. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics 12, 55–67.

Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. Neural networks 2, 359–366.

Hsu, J., Kalesnik, V., 2014. Finding smart beta in the factor zoo. Research Affiliates (July) .

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

Kingma, D. P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Kozak, S., Nagel, S., Santosh, S., 2017. Shrinking the cross section. Tech. rep., National Bureau of Economic Research.

Lecun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444.

Lettau, M., Ludvigson, S., 2001. Consumption, Aggregate Wealth, and Expected Stock Returns. The Journal of Finance 56, 815–849.

Masters, T., 1993. Practical Neural Network Recipes in C++. Morgan Kaufmann, google-Books-ID: 7Ez_Pq0sp2EC.

Ramachandran, P., Zoph, B., Le, Q. V., 2017. Searching for Activation Functions. arXiv:1710.05941 [cs] ArXiv: 1710.05941.

Rapach, D., Zhou, G., 2013. Forecasting Stock Returns. In: *Handbook of Economic Forecasting*, Elsevier, vol. 2, pp. 328–383.

Schwert, G. W., 2003. Anomalies and market efficiency. Handbook of the Economics of Finance 1, 939–974.

Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society: Series B (Methodological) 58, 267–288.

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

Zou, H., Hastie, T., 2005. Regularization and variable selection via the Elastic Net. Journal of the Royal Statistical Society, Series B 67, 301–320.