

Evaluation of Machine Learning in Empirical Asset Pricing

Ze Yu Zhong

Thursday 18th April, 2019

Todo list

<input type="checkbox"/> RS: Data skepticism (have you set up the data properly?) Show/highlight non-robustness of ML methods Skepticism of validity of methods that include non-stationary factors (like dividend yield) Check validity via simulation methods Methods to consider: OLS, Elasticnet, RF, NN See how it performs on real data	2
<input type="checkbox"/> RS: pending Literature Review	2
<input type="checkbox"/> RS: Pending	3
<input type="checkbox"/> RS: insert a picture of how the windows work from say caret documentation, would be great .	4
<input type="checkbox"/> RS: cite Huber loss	4
<input type="checkbox"/> RS: cite something about outliers in financial data	4
<input type="checkbox"/> RS: diagram representing huber loss would be great	4
<input type="checkbox"/> RS: update when algorithm is written	5
<input type="checkbox"/> RS: this diagram needs work	6
<input type="checkbox"/> RS: would be best to customize neuron text	6
<input type="checkbox"/> RS: justify properly	8
<input type="checkbox"/> RS: Pending	11
<input type="checkbox"/> RS: Pending	11
<input type="checkbox"/> RS: research different types of algorithms. Paper used ADAM	13

Contents

1 Introduction	2
1.1 Motivations	2
1.2 Literature	2
1.3 Main Findings	3
1.4 Limitations of Machine Learning	3
2 Methodology	3
2.1 Overall Model Design	3
2.2 Sample Splitting	3
2.3 Linear Model	4
2.3.1 Huber Loss	4
2.4 Penalized Linear Model	4
2.5 Classification and Regression Trees	5
2.6 Random Forests	6

2.7	Neural Networks	6
2.7.1	Introduction	6
2.7.2	Activation Function	7
2.7.3	Computation	8
2.7.4	Batch Normalization	8
2.8	Simulation Design	8
2.8.1	Overall Design	8
2.8.2	Simulating Characteristics	9
2.8.3	Simulating Macroeconomic Series	9
2.8.4	Simulating Return Series	10
2.8.5	Sample Splitting	10
2.9	Model Evaluation	10
2.9.1	R Squared	10
2.9.2	Diebold-Mariano Test	10
2.10	Variable Importance	11
3	Study	11
3.1	Data	11
3.2	Model	11
4	Appendix	11
4.1	Algorithms	11
4.1.1	Penalized Linear	11
4.1.2	Classification and Regression Trees	11
4.1.3	Random Forest	12
4.1.4	Neural Networks	13

1. Introduction

1.1. Motivations

The non-robustness has been

The inclusion of non-stationary factors such as dividend yield into more recent models is also of concern, as non-stationary factors by definition do not contain any information regarding the future.

To achieve this

Linear models, penalized linear models, random forests and neural networks were the machine learning algorithms considered.

1.2. Literature

Dramatic increase in financial factors, skepticism as to whether or not they are actually significant (factor zoo) Failure of traditional methods to perform in high dimensionality Difficulty of humans to interpret and find complex, non-linear interactions Machine learning, geared towards prediction Literature has shown that a lot of these techniques show promise, and in some papers, outperform quite consistently

Hsu and Kalesnik (2014)

Gu et al. (2018)

RS: Data skepticism (have you set up the data properly?) Show/highlight non-robustness of ML methods Skepticism of validity of methods that include non-stationary factors (like dividend yield) Check validity via simulation methods Methods to consider: OLS, Elasticnet, RF, NN See how it performs on real data

RS: pending Literature Review

1.4. Limitations of Machine Learning

Machine learning excels at prediction problems, namely estimating $E(r_{i,t+1}|\mathcal{F}_t)$, where $r_{i,t+1}$ is an asset's excess return over the risk free rate, and \mathcal{F}_t is the set of all information (including unobservable) available to market participants in this context.

This means that machine learning algorithms do not, nor do they aim to, explain how the market works in terms of underlying dynamics and equilibria. Though a machine learning algorithm may be able to identify patterns that otherwise cannot be easily found, an economist is still required to analyse these patterns to construct and hypothesize economic theory.

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 the model. Computational methods such as algorithms however, will only have their general principles and background explained in the appendix. This is because there are many variations of algorithms available, and more importantly, specific understanding of how the algorithm works is not necessary.

All asset excess returns are modelled as an additive prediction error model:

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

where

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

with $g^*(z_{i,t})$ representing the model approximation using the predictor set $z_{i,t}$.

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.

RS: insert a picture of how the windows work from say caret documentation, would be great

2.3. Linear Model

The least complex model considered is the simple linear regression model, otherwise known by its estimation method ordinary least squares (OLS). OLS struggles 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 :

$$g(z_{i,t}; \theta) = z'_{i,t} \theta \quad (3)$$

This model can capture non-linearities only if the predictor set $z_{i,t}^*$ contains specified non-linear transformations or interaction terms. It is quite common to consider at least second order terms and two way interactions.

The baseline computational algorithm for this model is to minimize the standard least squares function:

$$\text{content...} \quad (4)$$

2.3.1. Huber Loss

$$H(x; \xi) = \begin{cases} x^2, & \text{if } |x| \leq \xi; \\ 2\xi|x| - \xi^2, & \text{if } |x| > \xi \end{cases} \quad (5)$$

The Huber Loss metric offers a combination of mean squared error and mean absolute error, with errors smaller than the threshold ξ being mean squared error and errors larger than the threshold ξ being absolute error. This allows for the model fit to be less sensitive to outliers which are quite common in financial data.

RS: cite Huber loss

2.4. 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:

RS: cite something about outliers in financial data

$$\mathcal{L}(\theta; \cdot) = \underbrace{\mathcal{L}(\theta)}_{\text{Loss Function}} + \underbrace{\phi(\theta; \cdot)}_{\text{Penalty Term}} \quad (6)$$

RS: diagram representing huber loss would be great

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

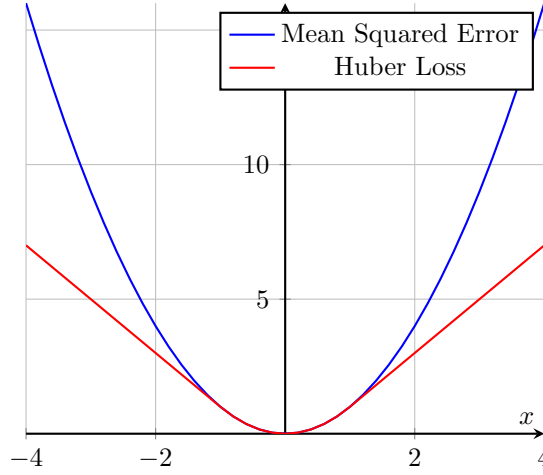


Figure 1. Illustration of MSE and Huber Loss when $\xi = 1$

$$\phi(\theta; \lambda, \rho) = \lambda(1 - \rho) \sum_{j=1}^P |\theta_j| + \frac{1}{2} \lambda \rho \sum_{j=1}^P \theta_j^2 \quad (7)$$

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 $\rho = 0$ case corresponds to the popular LASSO and uses absolute (l_1) parameter penalization, 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.

The $\rho = 1$ case corresponds to ridge regression, which uses l_2 that shrinks all coefficients closer to 0, but not actually 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 hyperparameters λ and ρ are both tuned using the validation sample. See appendix for algorithm.

RS: update when algorithm is written

2.5. Classification and Regression Trees

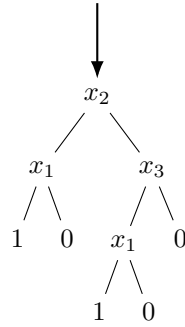
Classification and regression trees are fully non-parametric models that can capture complex multi-way interactions. A tree "grows" in a series of iterations. With each iteration, a split ("branch") is made along one predictor such that it is the best split available at that stage (in terms of lowering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

$$g(z_{i,t}; \theta, K, L) = \sum_{k=1}^K \theta_k \mathbf{1}_{z_{i,t} \in C_k(L)} \quad (8)$$

For this study, only recursive binary trees (the most common and easy to implement) are considered. The popular l_2 impurity was also chosen as the loss function (conceptually similar to mean squared error):

$$H(\theta, C) = \frac{1}{|C|} \sum_{z_{i,t} \in C} (r_{i,t+1} - \theta)^2 \quad (9)$$



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 is simply $\theta = \frac{1}{|C|} \sum_{z_{i_0,t} \in C} r_{i,t+1}$ i.e. the average of the partition.

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 to regularize trees by combining many different trees into a single prediction.

RS: this diagram needs work

2.6. 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. 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.

2.7. Neural Networks

2.7.1. Introduction

Neural networks are arguably the most complex type of model available, able to capture several non-linear interactions through 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.

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 in 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.

RS: would be best to customize neuron text

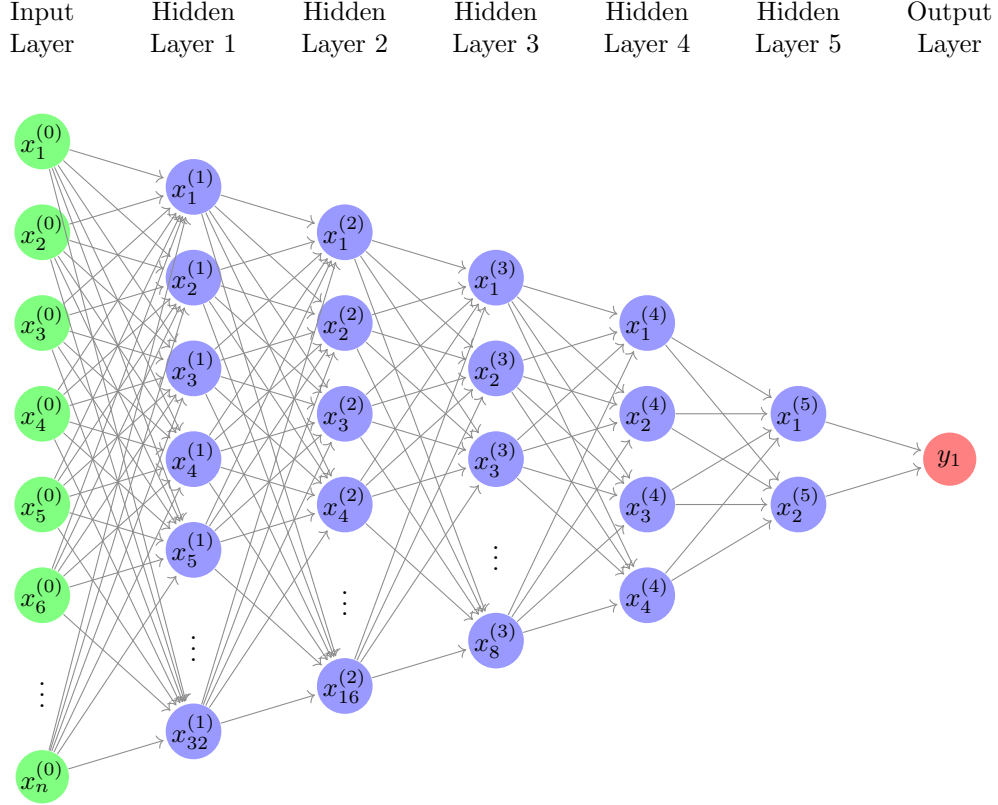


Figure 2. Neural Network 5 (most complex considered)

2.7.2. Activation Function

The ReLU activation function:

$$\text{ReLU}(x) = \max(0, x) \quad (10)$$

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)). Other potential choices for activation functions such as sigmoid, softmax, tanh etc. were not used due to computational cost.

The neural networks detailed in this paper have the following general formula. Let $K^{(l)}$ denote the number of neurons in each hidden layer $l = 0, 1, \dots, L$, with $l = 0$ denoting the input layer and $l = L$ denoting the output layer. Define the output of neuron k in layer l as $x_k^{(l)}$. Next, define the vector of outputs for this layer as $x^{(l)} = (1, x_1^{(l)}, \dots, x_{K^{(l)}}^{(l)})'$. The input layer is defined using predictors, $x^{(0)} = (1, z_1, \dots, z_N)'$. The recursive output formula for the neural network at each neuron in layer $l > 0$ is then:

$$x_k^{(l)} = \text{ReLU}(x^{(l-1)'} \theta_k^{l-1}), \quad (11)$$

with the final output

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

The neural network's weight and bias parameters are estimated by minimizing the penalized l_2 objective function of prediction errors.

2.7.3. Computation

The solution to finding the optimal weights and biases for the neural network is therefore found via the following chain rule:

$$\text{content} \dots \quad (13)$$

The solution to this is typically found via backpropagation, an iterative procedure similar to Gauss-Newton steps. 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 optimising the loss function with respect to the entire training sample, only a small, random subset of the data (mini batches) 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 learning rate (step size of each descent) needs to be shrunk towards zero as the gradient approaches zero to avoid the noisiness of the mini batch causing an "overshoot" of the optimum. A learning rate shrinkage algorithm was therefore employed.

2.7.4. 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.

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.

2.8. Simulation Design

2.8.1. Overall Design

One of my key concerns with the Gu et al. (2019) design is that the factors are uncorrelated across i , and, in particular, that the factors which do not matter in the return equation are uncorrelated with those that matter. This is not what is observed in practice.

Gu et al. (2018)

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

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

$$e_{i,t+1} = \exp\left(\frac{\sigma_{i,t+1}}{2}\right) \varepsilon_{i,t+1}, \quad (15)$$

$$\sigma_{i,t+1}^2 = \omega + \gamma_i \sigma_{t,i}^2 + w_{i,t+1}; \quad (\omega, \gamma, w) = (-0.736, 0.90, \sqrt{0.363}) \forall i \quad (16)$$

RS: justify properly

Let v_{t+1} be a 3×1 vector of errors, and $w_{i,t+1}, \varepsilon_{i,t+1}$ scalar error terms. 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.

2.8.2. Simulating Characteristics

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

$$\bar{c}_{ij,t} = \rho_j \bar{c}_{ij,t-1} + \epsilon_{ij,t}; \quad \rho_j \sim \mathcal{U}[1/2, 1] \quad (17)$$

Then, define the matrix

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

which we transform into a correlation matrix W via

$$W = \text{diag} \left(\frac{-1}{2} \right) (B) B \text{diag} \left(\frac{-1}{2} \right) (B) \quad (19)$$

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

$$\hat{C}_t = W \bar{C}_t \quad (20)$$

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

$$c_{ij,t} = \frac{2}{n+1} \text{rank}(\bar{c}_{ij,t}) - 1. \quad (21)$$

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

2.8.3. Simulating Macroeconomic Series

For simulation of x_t , a 3×1 multivariate time series, we consider a VAR model

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

where we have three separate specifications for the matrix A :

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

2.8.4. Simulating Return Series

We will consider four different functions $g(\cdot)$

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

$g_1(z_{i,t})$ allows the ch and $g_2(z_{i,t})$

Need to work out the corresponding cross-sectional R^2 in this case. We can then tune θ^0 to be this close to Gu et al. (2019), as well as the predictive R^2 . This will require some work.

Follow Gu et al. (2019) in regards to the choice of N, T, P_c

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.

2.8.5. 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.

2.9. Model Evaluation

2.9.1. R Squared

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

$$R_{OOS}^2 = 1 - \frac{\sum_{(i,t) \in \mathcal{T}_3} (r_{i,t+1} - \hat{r}_{i,t+1})}{\sum_{(i,t) \in \mathcal{T}_3} r_{i,t+1}^2} \quad (22)$$

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

2.9.2. Diebold-Mariano Test

The Diebold-Mariano test 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:

$$S_1^* = \left[\frac{n+1-2h+n^{-1}h(h-1)}{n} \right]^{1/2} S_1 \quad (23)$$

$$S_1 = \left[\hat{V}(\bar{d}) \right]^{-1/2} \bar{d} \quad (24)$$

$$\hat{\gamma}_k = n^{-1} \sum_{t=k+1}^n (d_t - \bar{d})(d_{t-k} - \bar{d}) \quad (25)$$

$$V(\bar{d}) \approx n^{-1} \left[\gamma_0 + 2 \sum_{k=1}^{h-1} \gamma_k \right] \quad (26)$$

2.10. Variable Importance

RS: Pending

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.

3. Study

3.1. Data

RS: Pending

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:

$$z_{i,t} = x_t \otimes c_{i,t} \quad (27)$$

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. Appendix

4.1. Algorithms

4.1.1. Penalized Linear

4.1.2. Classification and Regression Trees

Breiman (2017)

Algorithm 1: Classification and Regression Tree

Initialize ;

for d from 1 to L **do**

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

 For each feature $j = 1, 2, \dots, P$, 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

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

 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:

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

4.1.3. Random Forest

Breiman (2001)

Algorithm 2: Random Forest

for b from 1 to B **do**

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

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

4.1.4. Neural Networks

There are numerous stochastic gradient descent algorithms available.

Kingma and Ba (2014)

RS: research different types of algorithms. Paper used ADAM

Algorithm 3: Early stopping via validation

Initialize $j = 0$, $\epsilon = \infty$ and select the patience parameter p (max iterations)

while $j \leq p$ **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 4: Batch Normalization for one activation over one batch

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

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

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

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

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

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

References

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

Breiman, L., 2017. Classification and Regression Trees. Routledge.

- 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.
- 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.
- 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 .
- Lecun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444.
- 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.
- Zou, H., Hastie, T., 2005. Regularization and variable selection via the Elastic Net. Journal of the Royal Statistical Society, Series B 67, 301–320.