STEVENS
INSTITUTE OF TECHNOLOGY
1870

Taming the Factor Zoo: A Machine Learning Approach

FA 590: Statistical Learning
Final Project

Written By:
Andre Sealy

Peng Fu

Vincent Bin

Contents

Abstract

We performed a rigorous analysis of machine learning regression models for the general problem of predicting the equity risk premium of various asset prices without accounting for transaction cost and optimal drawdown strategies. Machine learning strategies have become more prevalent in asset pricing research; however, these strategies tend to create unnecessary bias and over-fitting. In addition to predicting equity risk premiums, we identify group-specific models that can reduce overfitting and improve performance.

Chapter 1

Introduction

Factor investing is a strategy that involves using quantifiable characteristics or ”factors” with the hope of achieving higher returns. The portfolio is weighted with different features than the classic other than classic market capitalization. The practice originates back to the early 1970s and have proven to achieve higher risk-adjusted returns than passive investment strategies in the past decade. Meaning, they offer higher risk premiums in excess of the market with factor premiums that are considered anomalies” in the Capital Asset Pricing model.

Over the past several decades, numerous empirical studies have documented these various anomalies in the Capital Asset Pricing Model (CAPM). By implementing these studies, we have found that financial characteristics such as low valuation multiples, high growth, or lower volatility achieve higher risk-adjusted returns than the model predicts over long periods. Over time, we have found other insightful factors to incorporate momentum and profitability, which are crucial for the growth of any publicly traded company. As such, CAPM was the first formal asset pricing model, and it provided the first precise definition of risk and how it drives expected returns.

Today, we have identified hundreds of factors in academic literature; however, only a few are generally accepted as adding incremental explanatory power. While there is some competition as to which is the best model, most practitioners tend to accept four or five-factor models, which include some combination of beta, size, profitability, and momentum. The problem only compounds when we consider the implementation of statistical learning in finance, as statistical models make it possible to run very complex, using a large amount of data, in very little time.

1.1 How Can Statistical Learning Help?

From the portfolio perspective of risk premium prediction, we are not necessarily interested in the accurate prediction of individual stock returns. Rather, we are interested in constructing a portfolio with good risk/return metrics, as the goal is to construct a portfolio to earn a risk-adjusted return. However, statistical learning only helps predict stock returns on the individual level, not the portfolio level.

The million-dollar question: Are methods that deliver the most accurate forecast at the individual stock level also provide the best-performing portfolio once we aggregate across stocks? This report utilizes statistical learning techniques that construct portfolios using a wide range of machine learning models and compare predicted performance to actual performance using financial risk metrics.

Chapter 2

Methodology

In this section, we will describe our collect of machine learning models that we use in our analysis, as well as the dataset, turning parameters and our choice of sample splitting and validation turning.

2.1 Data

Our analysis dataset comprises the monthly individual equity returns financial databases such as the Center for Research in Security Prices (CRSP) and Compustat, which list firms in the NYSE (possibly AMEX and NASDAQ). The sample begins in January 1957 and ends in November 2021, totaling 64 years. There are 32,655 in the sample, which has an average number of 5250 stocks per month.

Our original sample also contains a large collection of individual stock-level features and macroeconomic features. Of these 109 features, 28 are monthly, 61 are annual, and 13 are quarterly. The dataset also includes 74 dummy variables corresponding to the first two digits of the Standard Industrial Classification (SIC) codes, which we will use for cross-sectional comparisons.

In addition to the dataset from CRSP, we have also decided to include the Fama/French (FF) 3 research factors from the Kenneth French research repository. We plan to use the 3-Factor FF model for our machine learning benchmark. These factors include small-minus-big (SMB), high-minus-low (HML), and the market risk premium. Since the original dataset already includes the proxy for the risk-free rate, the 3-month Treasury bill, we have not included this feature from the Kenneth French Library. The dataset comes with a very important feature called size (mvel1), which is used as a proxy for market capitalization. For our analysis, we've only focused on stocks larger than a specific size. The reason is that micro-cap stocks are generally thought of as very speculative. Speculative assets tend to be very noisy, which can significantly impede predictability. Micro-cap stocks tend to be less

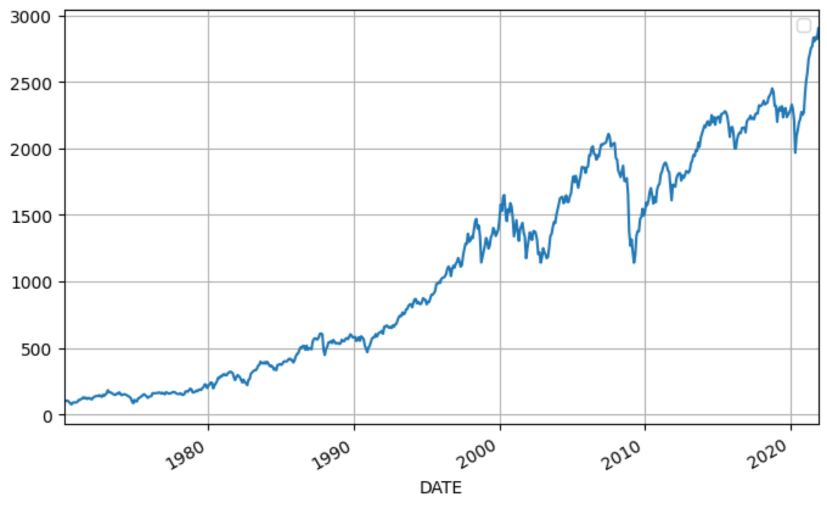


Figure 2.1: Historical number of stocks in our research period

established, which means they need attributes that may be important for forecasting, such as earnings, profits, capital expenditures, cash flow, and other financial metrics. This creates an issue where a significant portion of the data is missing and requires interpolation. To avoid this issue, we have decided to limit our research to stocks with a market capitalization of greater than $1 million dollar market capitalization.

Another technique we have used to avoid the interpolation issue is dropping features with more than 60 percent of its observations missing. This procedure impacts very few features such as Real Estate (realestate), which are the real estate holdings for a particular company in a given year. Our data cleaning procedure still resulted in a significant amount of missing values, which we replaced with the cross-sectional median at each month for each stock. From here, we still had a hand full of features with missing values: secured debt (secured), abnormal earnings announcement volume (aeavol), cash holdings (cash), change in tax expense (chtx), corporate investment (cinvest), earnings announcement return (ear), financial statement score (ms) ,number of earnings increase (nincr), return on assets (roaq), earnings volatility (roavol), return on equity (roeq), revenue surprise (rsup), accural volatility (stdacc), and cash flow volatility (stdcf). We have decided to drop these particular features.

The data cleaning process leaves us with 89 out of the initial 109 features, which gives us 681,960 observations out of the original ∼4 million observations. Our new dataset begins on Feburary 1965 and ends on November 2021, which ranges 682 months. This gives us a total of 7,722 unique stocks, down from 32,655 in the original dataset.

The number of stocks in our dataset changes every year, as new companies go public through initial publlic offerings and more establish companies delist themselves (either through private transactions or from regulatory/financial considerations). There is a monthly average of 1,000 stocks in our universe of stock, with a minimum of 72 and a maximum of

2,902. Figure (2.1) shows the number of publicly traded stocks increasing from the minimum of our dataset to the max.

It should be intuitive that our universe of stocks gradually increases over time, with some sharp declines due to historical periods of stock market crashes. Although there were periods of uncertainty in the late 1970s and 1980s, the 2000s were plagued with the 9/11 terrorist attack, the 2008 Financial Crisis, and the 2020 Pandemic, all of which created turmoil for financial markets. Considering these factors, we should expect our machine learning models to become unstable during this period as the composition of our universe of stocks changes.

2.2 Exploratory Data Analysis and Feature Engineering

When it comes to exploratory data analysis, there isn't a significant amount to explore. One can approach this problem by looking at the distribution of each feature of note. (we would find that the distribution of the majority of features is very skewed) The problem with this approach is that it is difficult to guess which features are more or less important than any other features. Secondly, the importance of the feature is dependent on the model and algorithm of choice.

Finally, our problem with begins the moment one decides to run a correlation matrix. As we can see from figure (**??**), very few assets tend to be correlated with the target variable. The variables with the strongest relationship with the risk premium is the market risk premium (macro_mtk-rf), stock variance (macro_svar), 3-month treasury bill (macro_tbl), earnings-to-price ratio of the S&P 500 (macro_ep), term spread (macro_tms), and net equity expansion (macro_ntis). It should come as no surprise that the equity risk premium has some significant amount of relationship with the macroeconomic features, considering the risk premium of a stock can be derived from some of these features. (interestingly, beta or beta squared have a significantly weaker relationship with the risk premium)

The right features are relevant to the task at hand and choosing the right features can be daunting as the model complexity increases. Normally, we should be able to handle this task with the implementation of feature engineering. Feature Engineering is the process of formulating the most appropriate features given the data, model and problem set; it's one of the most crucial task in any particular machine learning pipeline. For the project dataset, apart from scaling and cleaning, we have elected not to implement any feature engineering technique. The reason for this is the asymmetric temporal nature of many of the features. If we combine a quarterly feature with an annual feature, this new feature could potentially manipulate the hyperparameter tuning processes and create unnecessary bias in the model. Model complexity also varies with time, as we will demonstrate in a later section.

Another reason why we have decided not to implement feature engineering is that many of

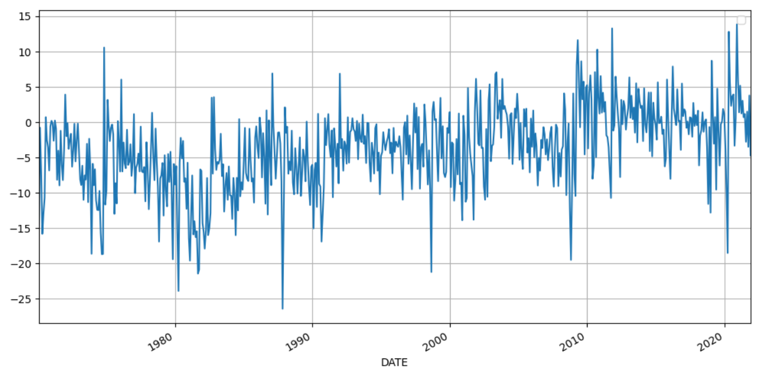


Figure 2.2: Historical risk premium

the features are well-known and have already been manipulated in some way. For example, we have equity performance metrics such as sales growth (sgr), return on equity (roeq), return on invested capital, etc. We also have many valuation metrics, such as earnings-to-price (ep), sales-to-price (sp), and book-to-market (bm). We do not need to manipulate these features because they are already a combination of other features, which may or may not already be present in our dataset. Utilizing feature engineering on any of these features would put us in a situation where we have a new feature that has little or no meaning and is not very significant to any of the machine-learning models.

The final reason not to use feature engineering is that most (if not all) of the features in our dataset are widely used in financial literature. For example, the dataset includes beta (beta) and beta squared (betasq), which is famously used by Eugene F. Fama and James D. MacBeth in their publication Risk, Return, and Equilibrium: Empirical Test. There are also momentum features, such as the 12-month momentum (mom12m), 1-month momentum (mom1m), 36-month momentum (mom36m), and the 6-month momentum (mom6m). Some of these features were utilized in Jegadeesh's research involving stock price mean revision seasonality and again in his 1993 research regarding stock market efficiency alone with Titamn [2][3]. As such, many of these features are already well defined for our problem set and there is no need to implement any feature engineering techniques.

2.3 Splitting and Validation

We have considered several sample-splitting techniques for forecasting; however, we have settled on the rolling window approach. The predominant asset price in quantitative finance is that stocks follow a geometric Brownian motion and are memoryless. That is, the stock price today is independent of the stock price at any period in the past. Since stock prices are memoryless, the underlying financial attributes may have little predictive power.

So, our approach involves using a window of 10 years for the training period, a 2-year window for the validation period, and a 1-year period for the testing period. This means we have a 10-year training sample (1965-1974), a 10-year validation sample (1975-1984), and the remaining 36 years (1985-2021) for out-of-sample testing. The splitting sample is rolled forward to include the most recent 12 months, and we refit the model every 12 months since most of our features are updated annually.

2.4 Standardization and Scaling

We have considered several standardization and scaling techniques for our dataset. In previous problems, we have used the Standard Scalar to scale the dataset; however, there are some issues that we could run into when implementing this technique for financial data. First, financial data is very susceptible to heavy tails and non-normal characteristics. Second, financial data often contains significant outliers, which may occur from the stock market crashes that are heavily prevalent in our dataset. Finally, financial data often exhibits temporal dependencies, which involve seasonality and statistical trends, and the standard scalar will ignore the time structure, which will distort temporal patterns.

Considering these factors, we have decided to implement a rank standardization between -1 and 1, based on the implementation of Kelly, Pruitt, and Su (2019) in modeling financial risk and return. The scaling method normalizes each feature by its cross-sectional standard deviation from each month. Then we map each of the features into the $[-1, 1]$ interval, thereby transforming features into a uniform distribution and decreasing the sensitivity to outliers. The raw results will be reported without any additional transformation or outlier extraction.

Chapter 3

Modeling

The following section outlines the statistical learning models that we have implemented in this report, a brief introduction to the function forms and how they work, and an outline of how they are implemented within our sample.

3.1 Simple Linear Regression Models

Our analysis begins with the least complex statistical model in our machine learning arsenal: the simple linear regression model. The functional form of a simple linear regression model is

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon \tag{3.1}$$

where y_i is the dependent variable, x is the independent variable, β_0 is the intercept, β_i is the coefficient of a given feature. We use this functional form as our benchmark for other statistical models, not because of the model's simplicity but because of another well-known financial model derived by linear regression: the Fama-French three-factor model. The following equation denotes the Fama-French three-factor model

$$r = R_f + \beta(R_m - R_f) + \beta_s \cdot \text{SMB} + \beta_v \cdot \text{HML} + \alpha, \tag{3.2}$$

where r is the portfolio's expected return, R_f is the risk-free rate, R_m is the return of the market portfolio, SMB is the difference between small and large companies (determined by market capitalization), and HML is the difference between high and low value companies (determined by the book-to-market ratio).[1]

[1]The general premise of Fama-French is that certain classes of stock tend to perform better than the market overall. Stocks of smaller companies tend to grow faster than stocks of more established companies. Stocks of undervalued companies tend to outperform stocks of overvalued companies. While this does not imply out-performance in today's market if one were to invest in these factors, it does show that empirically these factors tend to perform well.

Modeling Process: Simple Linear Regression

Using the data from the Kenneth French research repository, we created another two macro variables to replicate the Fama-French three-factor model. Afterward, we recreated the same model using features from the original dataset, which was then adapted to add 7 and 15 additional features[2]. Afterward, we ran the simple linear regression model to incorporate all of the features in the dataset.

3.2 Residual Minimizing Models

The simple linear regression and the Fama-French model derived from it are some of the most powerful tools in finance. However, there are many obvious problems with using the simple linear regression model, one of which involves the large amount of bias involved with a linear model[3]. One tool we used to minimize this bias is utilizing the Huber Regression.

The Huber loss function is an extension of the simple linear regression, which utilizes mean squared error (MSE) and mean absolute error (MAE) to handle outliers for regression tasks. It adjusts how residuals are penalized based on their magnitude by using the MSE (squared loss) to handle small residuals and the MAE (absolute loss) to deal with more significant residuals. The functional form for the Huber loss function is defined as:

$$L(u) = \begin{cases} \frac{1}{2}u^2 & \text{if } |u| \leq \delta \\ \delta \cdot |u| - \frac{1}{2}\delta^2 & \text{if } |u| > \delta \end{cases} \tag{3.3}$$

where $u = y - \hat{y}$ and δ is the threshold parameter that determines the boundary between squared and absolute loss. Thus, the goal of the Huber Regression is to minimize the following:

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \sum_{i=1}^{n} L\big(y_i - (\beta_0 + \beta_1 x_{i1} + \beta 2 x_{i2} + \cdots + \beta_p x_{ip})\big) \tag{3.4}$$

where $L(\cdot)$ is the Huber loss function as defined above. The general intuition behind this model is that the MSE penalizes large residuals heavily due to squaring, which makes it sensitive to outliers, while the MAE treats all residuals linearly, reducing sensitivity to outliers. This model combines the advantages of penalizing smaller residuals quadratically for smoother optimization while penalizing more significant residuals linearly to reduce outlier impact.

[2]The benchmark recommendations were derived from Lewellen (2015), which recommended either 3, 7, or 15 features.

[3]The linear regression model assumes that the observations follow a linear pattern, when it could be non-linear

3.2.1 Modeling Process: Huber Regression

For the Huber loss function, we implemented the same modeling pipeline: 1) using the Fama-French three-factor model, 2) constructing a synthetic Fama-French three-factor model, 3) adapting to 7 and 15 additional factors, and 4) using all of the features in our dataset. After this process, we compare the performance of the Huber loss function to our benchmark portfolio.

3.3 Regularization Models

The Huber Regression is a formidable expansion of the simple linear regression model, and it appears to perform better than the simple linear regression when it comes to more manageable datasets. However, it comes to datasets where p is significantly large, model complexity increases, which increases the likelihood of overfitting th model. As such, it can become difficult to achieve additional performance benefits from penalizing residuals alone. To reduce overfitting the model, we introduce the Elastic Net model.

The Elastic Net model is a type of linear regression that combines two regularization techniques: the Lasso regression (L1) and Ridge regression (L2) to address the limitations of each. The objective of the Lasso regression is the following:

$$\min_{\beta}\left\{\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} \quad \text{subject to} \quad \sum_{j=1}^{p}|\beta_j| \leq s. \tag{3.5}$$

Similarly, the objective of the Ridge regression is the following:

$$\min_{\beta}\left\{\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} \quad \text{subject to} \quad \sum_{j=1}^{p}\beta_j^2 \leq s. \tag{3.6}$$

The Elastic Net combines the objectives of equation (3.5) and equation (3.6) to provide the following objective

$$\min_{\beta}\left\{\frac{1}{2n}\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda_1||\beta||_1+\lambda_2||\beta||_2^2\right\} \tag{3.7}$$

where λ_1 is the Lasso penalty, λ_2 is the Ridge penalty, $||\beta||_1$ is the L1 norm for feature selection and $||\beta||_2^2$ is the L2 norm for shrinkage. In other words, unlike the simple linear and the Huber regression, the Elastic Net allows us to tune the parameters to improve performance. The obvious downsides of finding the optimal parameters are the computational considerations.

3.3.1 Modeling Process: Elastic Net

We've decided to use the default L1 parameter of 0.5; the L2 parameter ranges between 10^{-4} and 10^{-1}. The process involves splitting the data into it's respective training, validation and testing sets, where we fit the model using the training and validation set using a given $\lambda \in [10^{-4}, 10^{-1}]$. Using the MSE as our scoring criteria, we then select λ that generates the lowest MSE for our given validation window. This parameter is then used to fit the model to predict the R^2_{OOS}.

3.4 Dimensionality Reduction Models

As mentioned previously, regularization's benefits can help select a model that prevents overfitting and reduces multicollinearity by shrinking or forcing coefficients near or exactly zero, leading to better performance. However, this can reduce the predictability of models when many of the features are correlated. Rather than allowing an algorithm to select suboptimal features, we can reduce the number of features of the model altogether. Two models that allow us to achieve this goal are Principal Components Regression and Partial Least Squares Regression .

3.4.1 Principal Components Regression

The Principal Components Regression (PCR) transforms features into a set of orthogonal components (principal components) using principal component analysis (PCA). We then use these principal components to run a linear regression using a subset of these components instead of the original features of the dataset. Recall the functional form of the simple linear regression model in equation (3.1). Instead of using the predictors, x_i, we use the principal components Z_k, which gives us the following:

$$y = \beta_0 + \beta_1 Z_1 + \beta_2 Z_2 + \cdots + \beta_m Z_m + \epsilon. \tag{3.8}$$

We choose m based on cross-validation, which is explained later in the modeling process for PCR. Since the principal components, Z_k are linear combinations of the original predictor, x_i, the regression can be expressed in terms of the original predictors:

$$y = \beta_0 + \sum_{k=1}^{m} \beta_k \left(\sum_{j=1}^{p} \phi_{kj} x_j \right) + \epsilon, \tag{3.9}$$

or equivalently:

$$y = \beta_0 + \sum_{j=1}^{p} \left(\sum_{k=1}^{m} \beta_k \phi_{kj} \right) x_j + \epsilon. \tag{3.10}$$

So the functional form shows that the PCR is essentially a simple linear regression model, but the coefficients are indirectly determined by the selected principal components.

Modeling Process: Principal Component Analysis

The modeling process for the PCR follows the same flow as the modeling process for the Elastic Net regression. The hyperparameter for the PCR is merely the number of components: $K = 50$. We use a list of predefined components between 1 and 50 and the MSE as our scoring metric. As previously stated, we select our K that minimizes the MSE and use the same parameter to predict the testing set.

3.4.2 Partial Least Square Regression

In many ways the Partial Least Squares (PLS) is similar to the PCR, in the sense that it reduces the dimensionality of features into latent components. However, unlike the PCA, it incorporates information from the labels while constructing these components, ensuring that the components are predictive. Rather than using features x_i and the labels y, we use a matrix predictor X and matrix response variable Y. Using these matrices, the PLS regression seeks to decompose X and Y into latent components:

$$X = TP^T + E, \quad Y = UQ^T + F, \tag{3.11}$$

where T is the score matrix for X, P^T are the loading matrices for X, U is the score matrix for Y, Q^T is the loading matrix for Y, and E and F are the residuals for X and Y, respectively.

This process involves standardizing X and Y, such that $X, Y \sim (0, 1)$. Then the PLS regression will find linear combinations of the predictors ($T = XW$) that can explain the variance in X and are optimally correlated with Y. We decompose X and Y into components representing the underlying relationship between predictors and responses.

Modeling Process: Partial Least Squares

The process is similar to the modeling process for PCR, where our choice of $K = 35$ was made after running the PLS regression on the entire dataset and recording the optimal number of components, which was found to be 35.

3.5 Gradient Boosted Regression Trees

Gradient Boosted Regression Trees (GBRT) is a statistical learning technique for performing regression tasks that combines two powerful algorithms: decision trees and gradient boosting. The combination of two or more learners is what practitioners call an ensemble learning method. The ensemble method in GBRT takes shallow decision trees, where each successive tree corrects the errors of the previous tree, which optimizes the loss function.

The functional form for GBRT can be expressed as follows:

$$F(x) = \sum_{m=1}^{M} \alpha_m h_m(x) \tag{3.12}$$

where $F(x)$ is the final predictive model based on the weighted sum of individual decision trees, M is the total number of trees in the ensemble, α_m is the learning rate, $h_m(x)$ are the individual trees that are trained to correct the residuals from previous trees, and x is the feature in the dataset.

The algorithm constructs the ensemble iteratively, where at each iteration m, the model is updated as: Where $F_{m-1}(x)$ is the model at iteration $m-1$, and $h_m(x)$ is the new decision tree that is trained on the negative gradient, with the loss function:

$$r_i^{(m)} = -\frac{\partial L\left(y_i, F_{m-1}(x_i)\right)}{\partial F_{m-1}(x_i)} \tag{3.13}$$

where here, $r_i^{(m)}$ are the residuals at iteration m, $L(y, \hat{y})$ is the loss function, y_i is the target value, and $F_{m-1}(x_i)$ is the prediction from the model at iteration $m-1$.

$$F_m(x) = F_{m-1}(x) + \alpha_m h_{m(x)} \tag{3.14}$$

Modeling Process: Gradient Boosted Regression Trees

GBRT is by far the most computationally expensive algorithm in our toolbox. Each computation on the full dataset can take anywhere from 7 hours to several days, even without the default parameters. As such, we took extra care to make sure that we could run all of the analytics we needed in a timely fashion without the quality of results. For example, we've implemented a maximum no change iteration parameter of 3 iterations. This parameter stops the algorithm early if there is no change in the validation output.

We've also minimized the number of potential parameters for the model. We've used a maximum depth of 2; the number of trees (estimators) ranges between 100 and 1000, and the learning rate ranges between 0.1 and 0.01. We've also implemented the Huber loss function to handle outliers in the validation step. These parameters will be tuned during the training and validation window, which will then be used to predict on the testing window.

3.6 Neural Networks

We have also decided to incorporate nonlinear models; the first of which involves the artificial Neural Network. The Neural Network (NN) is arguably the most powerful machine learning model, as most practitioners tend to use NN over other complex models. The model is inspired by the way biological neural systems work, and designed to learn patterns and relationships in data. It consist of layers of interconnected units called neurons. These neurons process input data through a series of transformations and activators. The general functional form for the NN is denoted by

$$\hat{y} = f_L \circ f_{L-1} \circ \cdots \circ f_1(x) \tag{3.15}$$

where x is the input vector, mapped to $x \in \mathbb{R}^n$, $\hat{y}$ is the output of the network, mapped $\hat{y} \in \mathbb{R}^m$, f_l is the function applied at layer l, and $\circ$ is the composition operator, indicating that the output of one layer is the input to the next. A more detailed formulation involves the input layer:

$$a^0 = x \tag{3.16}$$

where x is the raw input vector. The hidden layers are denoted by:

$$\begin{aligned} z^{(l)} &= W^{(l)}a^{(l-1)} + b^{(l)}, \quad \text{(Linear transformation)} \\ a^{(l)} &= \sigma^{(l)}\left(z^{(l)}\right), \quad \text{(Non-linear actication)} \end{aligned} \tag{3.17}$$

Finally, the output layer L, denoted by:

$$z^{(L)} = W^{(L)}a^{(L-1)} + b^{(L)}, \quad \text{(Linear transformation)} \tag{3.18}$$

where $\hat{y} = \sigma^{(L)}\left(z^{(L)}\right)$ is the output activation. There are dozens of activation functions, $\sigma(z)$, for the NN, but the most common choices include the Rectified Linear Unit (ReLU), denoted by $\max(0, z)$, the Sigmoid function, denoted by $\frac{1}{1+e^{-z}}$, the Tanh function, denoted by $\frac{e^z - e^{-z}}{e^z + e^{-z}}$, and Softmax for classification task (which we won't outline, since this is a regression problem). In general, we have the functional form of the NN

$$\hat{y} = \sigma^{(L)}\left(W^{(L)} \cdots \sigma^{(1)}\left(W^{(1)}x + b^{(1)}\right) \cdots + b^{(L)}\right) \tag{3.19}$$

Modeling Process: Neural Networks

The NN model utilizes the largest hyperparameters; however, we only tune two specific parameters: the L1 penalty (similar to the Elastic Net) and the Learning Rate. We set the L1 parameter λ between the interval 10^{-5} and 10^{-3}, and the learning rate is a choice between 0.001 and 0.01. These are the only hyperparameters that are variable; the rest are constant. Our constant parameters allow the algorithm to run efficiently and quickly. These hyperparameters include the batch size, epochs, patience, and ensemble. The most important hyperparameter is the activation function. We have decided to use the ReLU function. The ReLU function allows for faster evaluation time for the number of active neurons.

The neurons are another hyperparameter that we hold constant in our model evaluation. As part of our evaluation process, we tune the same hyperparameters while changing the number of neurons in the hidden layers. We first start with a single hidden layer of 32 neurons, which we denoted as NN-1. The second model, NN-2, has two hidden layers with 32 and 16 neurons, respectively; NN-3 has three hidden layers with 32, 16, and 8 neurons; NN-4 has four layers with 32, 16, 8, and 4 neurons; and NN-5 has five hidden layers with 32, 16, 8, 4, and 2 neurons.

The idea behind selecting the hidden layers in this fashion stems from using a funnel architecture following a binary decimal system, a common approach in deep learning. The first layer (32 neurons) will attack as the initial layer in the neural network, where it will learn the raw input of the data; the middle layers (16, 8 neurons) will learn the more abstract representation of the input data; and the final layers (4, 2 neurons) are distilled to the essential information required to make predictions. The benefits of this structure are mainly for efficiency, as it drastically improves the computation time; however, we also receive the benefits of a reduction in overfitting and generalizing model predictability.

3.7 Hyperparameter Tuning Overview

Table 3.1: Hyperparmeters for All Models

	Linear	Huber	PLS	PCR	ENet	GBRT	NN1-NN5
Parameters	None	None	K	K	$\lambda \in (10^{-4}, 10^{-1})$ L1=0.5	Depth=1 or 2 Loss Function=Huber LR$\in \{0.01, 0.1\}$ Trees=$\{100, 300, 500, 1000\}$	$\lambda_1 \in (10^{-5}, 10^{-3})$ LR$\in \{0.001, 0.1\}$ Batch Size=5,000 Solver=Adam Para Activation=ReLU

Table (3.1) shows each of the models in our evaluation process, along with their respective hyperparameters, outlined from the least amount of parameters to the greatest. As we can see, the NN1-NN3 models have the most hyperparameters, which means these models employ the least amount of assumptions to fit the functional form to the dataset.

Chapter 4

Results

This chapter will outline the results for both simple and complex models. Ultimately, the difference between our classification of "simple" and "complex" models stems from whether or not the model requires hyperparameters, which must be set before training the model to guide the learning process. The Huber and simple linear regressions are the only two models in our research that do not require this process.

4.1 Simple Models

This section will outline the cross-sectional out-of-sample predictive R^2 (denoted by R^2_{oos}). We compare eight models which are de (OLS-FF, OLS-3, OLS-5, OLS-7, OLS-15, LR-FF, LR-3, LR-5, LR-7, LR-15). Afterward, we then use the R^2_{oos} to construct statistical learning portfolios and compare overall predictability performance using common statistical, financial metrics. Finally, we selected the best models and assessed the most impactful features for overall performance.

4.1.1 Simple out-of-Sample R^2 model comparison

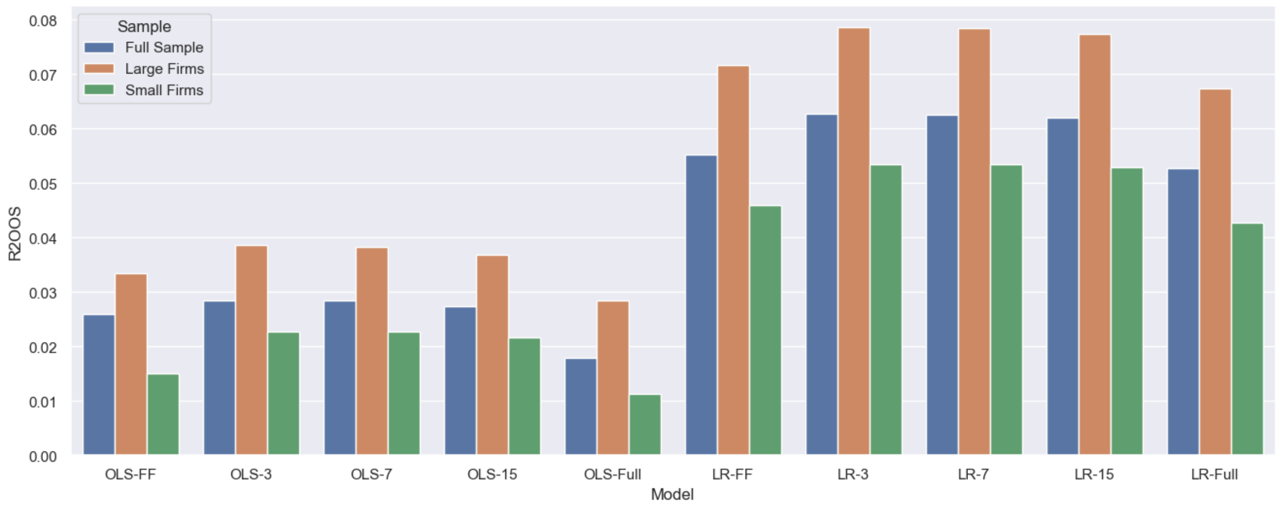
Table (4.1) reports the annual R^2_{oos} for the dataset using the simple linear regression and OLS. LR-FF and OLS-FF utilize the Fama-French three-factor model, while LR-3 and OLS-3 utilize the synthetic Fama-French three-factor model. Models such as LR-7, LR-15, OLS-7, and OLS-15 are extensions of the Fama-French three-factor model[1]. The final two months, LR-Full and OLS-Full, utilize all of the variables in our dataset. In addition to recording the R^2_{oos} for the entire dataset, we have also recorded the results for the top and bottom 30% of stocks, based on their market capitalization.

[1]Our Fama-French models use small-minus-big, high-minus-low, the market risk premium, and the three-month treasury bill as its factors. The OLS-3/LR-3 uses 12-month momentum, size, and book-to-market. OLS-7/LR-7 uses OLS-3/LR-3 plus working capital accurals, sales growth, asset growth, and growth in common shareholder equity. OLS-15/LR-15 uses OLS-7/LR-7 plus dividends to price, 36-month momentum, beta, return volatility, share turnover, leverage, and sales to price.

As we can see, the simple linear regression post a lower R^2_{oos} than the Huber regression for all models in our analysis. This should not be surprising, as the Huber regression typically produces a lower R^2 due to the way it handles outliers and its objective function, as we've explained in Chapter 4. For the simple linear regression, if the data has outliers, the model may fit these outliers well, which gives a higher R^2_{oos}. However, the Huber regression breaks down the influence of these outliers, and the model does not fit them as well, leading to potentially higher residuals. Since our R^2_{oos} depends on residuals, if the model does not fit the outliers well, it may result in a lower R^2_{oos}

Table 4.1: Annual out-of-sample risk-premium predictive performance (R^2_{oos}): simple models

	OLS-FF	OLS-3	OLS-7	OLS-15	OLS-Full	LR-FF	LR-3	LR-7	LR-15	LR-Full
Full Set	2.60	2.85	2.84	2.74	1.79	5.52	6.27	6.26	6.21	5.27
Top 30%	3.35	3.86	3.83	3.69	2.85	7.16	7.86	7.84	7.74	6.73
Bottom 30%	1.51	2.27	2.27	2.17	1.12	4.59	5.35	5.35	5.30	4.28



As we can also see from the table, the model improves the more factors we introduce; however, we do experience diminishing returns, especially with the simple linear regression. The model peaks in performance with the LR-7, which has an R^2_{oos}, but then drops when more features are included. This trait is shared with the Huber regression as well, regardless of the size of the stocks or the focus of the industry in question.

However, recall that we have designated the Fama-French models as our benchmark for our R^2_{oos}, which means, at minimum, we expect most models to perform similar, if not better, than our initial benchmark. We can see that most models perform better than the Fama-French models, except for the OLS and LR regression with all the covariates in our dataset.

4.1.2 Statistical Learning Portfolios: Simple Models

After building the simple linear and Huber regressions for each pre-selected portfolio, We construct a statistical learning portfolio by selecting the top 100 stocks with the highest $\hat{y}$. The top 100 stocks are used to generate an equally weighted portfolio, denoted by

$$r_{p,t} = \frac{1}{100}\sum_{i=1}^{100} r_{i,t}. \tag{4.1}$$

This equally weighted portfolio is generated for each month, T, which is then used to compute the average portfolio risk-premium, denoted by

$$\bar{r}_p = \frac{1}{T}\sum_{t=1}^{T} r_{t,p} \tag{4.2}$$

This will serve as our basis for analyzing the accuracy and performance of the statistical learning portfolio. The optimal prediction algorithm can precisely predict the actual returns in an out-of-sample setting.

Table 4.2: Performance of the statistical learning portfolios

	Actual Returns $\bar{r}_p$	Predicted Returns $\hat{\bar{r}}_p$	Volatility σ_r	Sharpe ratio
LR-FF	-3.54%	-4.75%	3.12%	-5.28%
LR-3	-3.19%	-4.07%	2.61%	-5.40%
LR-7	-3.34%	-4.01%	2.62%	-5.29%
LR-15	-3.28%	-3.79%	2.66%	-4.94%
LR-Full	-3.26%	-3.79%	3.21%	-4.09%
OLS-FF	-3.54%	-4.60%	3.34%	-4.77%
OLS-3	-3.16%	-4.05%	2.62%	-5.36%
OLS-3	-3.16%	-4.05%	2.62%	-5.36%
OLS-7	-3.26%	-4.02%	2.61%	-5.34%
OLS-15	-3.30%	-3.81%	2.69%	-4.90%
OLS-Full	-3.28%	-3.79%	2.66%	-4.94%

Table (4.2) shows the performance of the statistical learning portfolios specifically for the Huber and simple learning regression. All ten models mentioned in the previous section are outlined, starting from the benchmark models and increasing incrementally to more complex models. The statistical learning portfolios are presented in terms of the algorithm/model of choice, the actual average portfolio returns, $\bar{r}_p$, the predicted average portfolio returns, $\bar{\hat{y}}_p$, the volatility, which is denoted by

$$\sigma_r = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}\left(r_{t,p} - \bar{r}_p\right)^2} \tag{4.3}$$

and the sharpe ratio, denoted by $\frac{\bar{r}_p}{\sigma_r}$.

Recall from Table (4.1) that the models with the best R^2_{oos} were the models with the smallest number of features. However, we can see that the models with the least features tend to be the least accurate. Note that both the LR and OLS Fama-French models should generate the same actual return of -3.54%; however, the Huber regression performs slightly better for the reasons we have outlined earlier.

As we can see, the more covariates we add to the model, the better it performs. The model with the best performance appears to be the OLS-Full, which uses all of the covariates. However, the LR-Full also uses all of the covariance and performs slightly worse but generates a higher Sharpe ratio (the portfolio is riskier but generates a higher return per unit of risk). Regarding predictability and performance, the Linear Regression model with all features performs the best out of the simpler models.

4.1.3 Feature Importance: Simple Models

It should come as no surprise that the more features introduced to linear models, the more their performance deteriorates. This can happen either through overfitting or multicollinearity. Although two models may perform the same (or better), the model with the most significant explanatory power and interpretability is better. We conducted feature importance analysis on the full covariate simple linear and Huber regression to quantify the interpretability. Feature importance refers to the contribution of each feature variable in a model to pre-

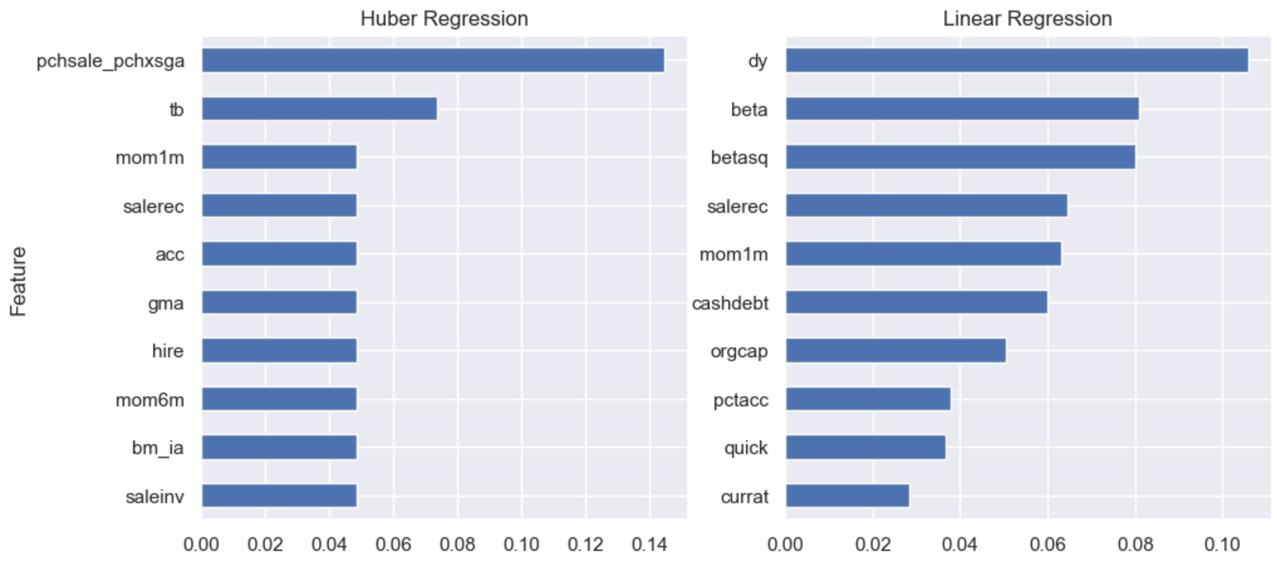


Figure 4.1: Top 10 important features of the Huber and Simple Linear Regression

dicting the target variable. This can help us identify the most impactful variables, guiding model selection and interpretation. Figure (4.1.3) shows the top ten important features for both the Huber and simple linear regression models with all covariates.

To compute the importance of each feature, we used a five-year training window (2013-2018) and a two-year validation window (2019-2021) to quantify the importance of each feature. This time interval is due to the performance breakdown of many of our statistical learning models within this interval. The process involves training an arbitrary model on the testing set, with the optimal set of hyperparameters, and then making predictions on the validation set, similar to any other machine learning pipeline. Afterward, we ran an algorithm that computes the R^2 for each feature individually, which we can use to find the marginal change in the R^2 per feature.

This works due to the definition of R^2 as $1 - RSS/TSS$, where $\sum (y_i - \hat{y})^2$ is the total sum of squares for the response. Since RSS always decreases as more variables are added to the model, the R^2 always increases as more variables are added. As such, we can find the variable importance by simply taking the difference in the overall R^2 and the marginal R^2 per feature.

The simple linear regression model shows that the most important features are dividend to price, beta, beta squared, sales to receivables, 1-month momentum, cash flow to debt, organizational capital, percent accruals, quick ratio, and current ratio. We receive different features regarding what is important for the Huber regression. The most important features for the Huber regression are the percentage change in sales minus the percentage change in SG&A, tax income to book income, 1-month momentum, sales to receivables, working capital accruals, gross profitability, employee growth rate, 6-month momentum, industry adjusted book to market, and sales to inventory. **Add more here potentially**

4.2 Complex Models

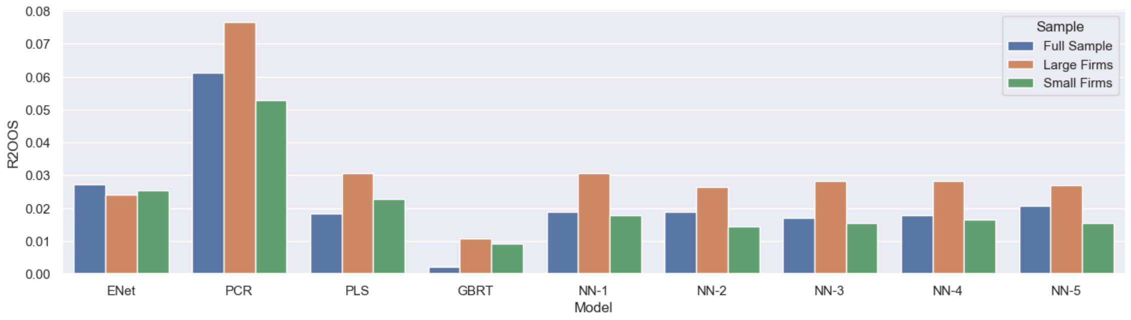
This section will outline the cross-sectional out-of-sample predictive R^2 (denoted by R^2_{oos}). We compare eight models in this section: Elastic Net, Principal Component Regression, Partial Least Squares Regression, and Neural Networks 1-5. Similar to the simple models, we then use the R^2_{oos} to construct statistical learning portfolios, compare overall predictability performance using common statistical and financial metrics, and compare feature importance for the models in each learning category. Finally, we examined how each model adjusts their overall tuning parameters over time, as the respective models change how they learn when more data is introduced to the algorithm.

4.2.1 Complex out-of-Sample R^2 model comparison

Compared to the simple models, we receive a wide range of results for complex models. What instantly stands out among the rest is the performance for the principal component regression; when modeling the full dataset, the PCR achieves an R^2_{oos} of 6.12, while all of the other models are under 3% percent in terms of performance. This may be because, in essence, the PCR does not require any hyperparameter tuning in the same way other models, such as the neural networks and elastic nets. The algorithm decides how many principal components to retain based on cross-validation. Regardless, it is still worth noting that the PCR performs the best out of all cross-sections.

Table 4.3: Annual out-of-sample risk-premium predictive performance (R^2_{oos}): complex models

	ENet	PCR	PLS	GBRT	NN-1	NN-2	NN-3	NN-4	NN-5
Full Set	2.74	6.12	1.82	0.02	1.89	1.90	1.97	1.79	2.06
Top 30%	2.42	7.66	3.06	1.06	3.05	2.64	2.83	2.82	2.69
Bottom 30%	2.54	5.27	2.26	0.09	1.77	1.46	1.54	1.64	1.56



The ENet and PLS model performs consistently well. Unfortunately, by far, the model with the worst out-of-sample performance is the Gradient Boosted Regression Trees, despite the limited number of parameters used for training, barely scratching a whole percentage point. This performance is disappointing, considering it's an ensemble model; however, complexity only sometimes ensures better performance or results.

The neural network models perform consistently better based on the number of hidden layers and neurons. The best-performing model in the set is PLS for the full dataset. However, for other cross-sections, the model neural network performs less consistently. This may be due to the lack of available observations in the raw input vector used for each hidden layer.

4.2.2 Statistical Learning Portfolios: Complex Models

Using equations (4.1), (4.2), and (4.3), we derive the output in Table (4.4), which shows the performance of statistical learning portfolios for the more complex models. The models that seems to match the actual returns fairly well are the PLS, PCR, ENet and NN-1 models. As expected, the neural network models perform better the more hidden layers and neurons are added to the model. We can see that there is a greater degree of consistency and performance when implementing models with hyperparameters.

Table 4.4: Performance of the statistical learning portfolios (Complex Models)

	Actual Returns $\bar{r}_p$	Predicted Returns $\hat{\bar{r}}_p$	Volatility σ_r	Sharpe ratio
Elastic Net	-3.23%	-3.81%	2.64%	-5.00%
PLS	-3.31%	-3.45%	3.06%	-3.90%
PCR	-3.39%	-3.89%	2.70%	-5.00%
GBRT	-3.57%	-4.48%	3.64%	-4.17%
NN-1	-3.40%	-1.74%	2.98%	-2.02%
NN-2	-3.43%	-1.92%	2.89%	-2.31%
NN-3	-3.48%	-1.84%	2.88%	-2.21%
NN-4	-3.46%	-2.13%	2.62%	-2.81%
NN-5	-3.47%	-2.74%	2.37%	-4.00%

Based on predicted performance compared to actual performance, we would say that the PLS model performs the best compared to the rest of the complex models. However, regarding the financial metrics, NN-2 performs the best on a risk-adjusted basis.

4.2.3 Feature Importance: Complex Models

Figure (4.2.3) shows the feature importance for the more complex models. The same process for computing feature importance for the simple models were also implemented for the more complex models.

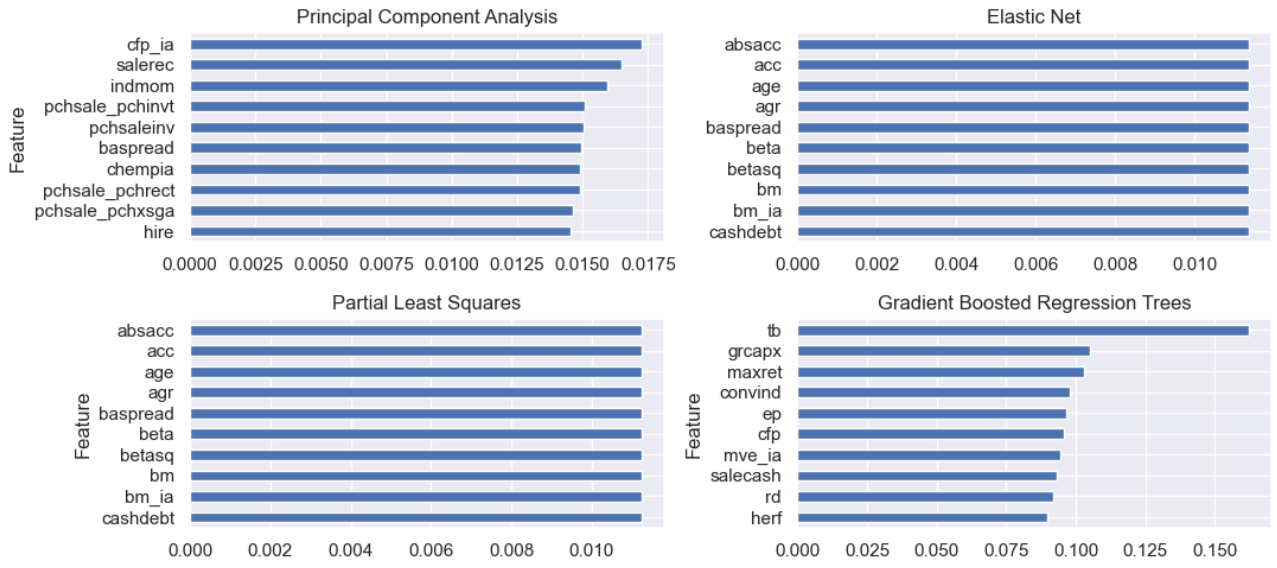


Figure 4.2: Time-varying model complexity

4.2.4 Time-Varying Model Complexity

In order to get a better idea of how our statistical learning models adjust and learn over time, we have conducted a time-varying analysis for PLS, PCR, ENet, and GBRT. The premise behind this analysis is that the complexity of the model is adjusted dynamically over time during the training and evaluation phase. This helps us to improve performance, reduce overfitting, and adapt to the changing distribution of the data.

To conduct this analysis, we merely record the optimal parameters for each rolling window for each machine-learning model in our time interval. Figure (4.2.4) outlines the parameters we have used for the four models in question. For ENet, we recorded the number of coefficients utilized when the L2 parameter, λ provided the optimal MSE for each training and validation window; for GBRT, we recorded the max depth for each training and validation window; and for PLS and PCR, we merely recorded the components K that provided the optimal MSE for each training and validation window. The minimum number of components for the PLS and PCR models is 5, while the maximum number of components is 35 and 50, respectively. The highest depth allowed for the GBRT is 2. As always, ENet shrinks the coefficients (potentially to zero) on the shrinkage parameter, λ, that provides the optimal MSE.

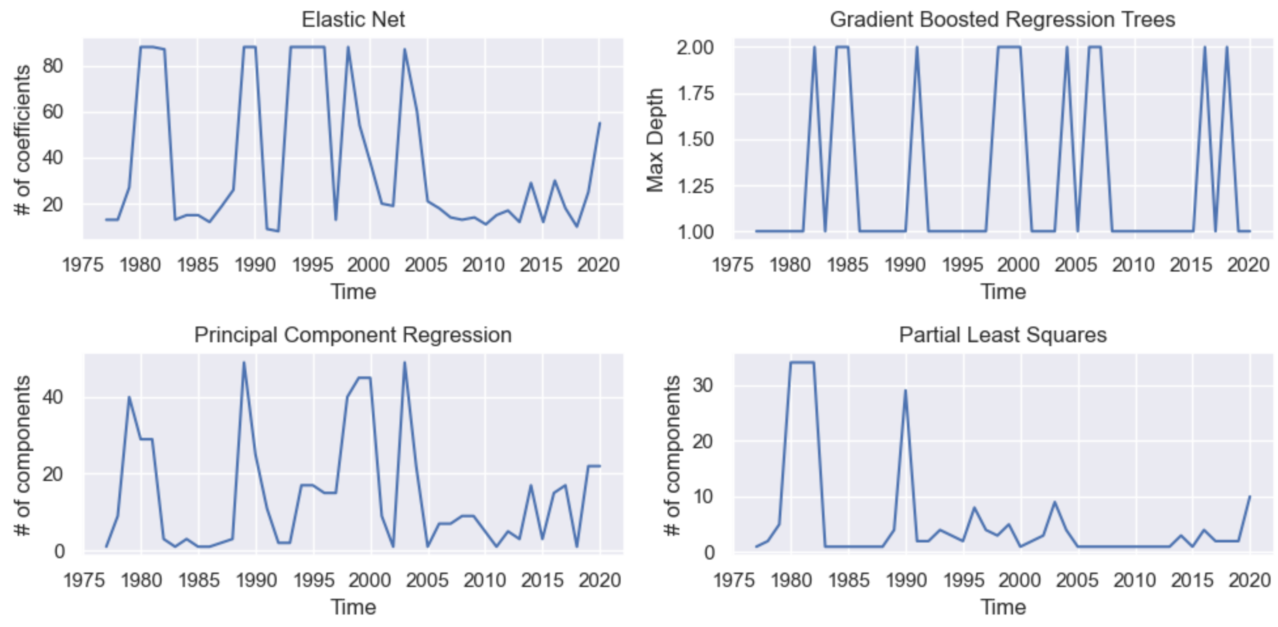


Figure 4.3: Time-varying model complexity

The time-varying complexity is relatively consistent across the board. From the late 1970s to the early 1980s, from the late 1980s to the early 1990s, from the late 1990s to the early 2000s, in the period of the Financial Crisis, and onward after 2015, the models increase in complexity through these specific eras in the last 45 years. The more features the model requires, the more complex the model becomes.

As the figure above shows, the degree of complexity varies based on the model. For the Elastic Net model, as we can see, when the Lasso penalty λ_1 dominates, the model becomes more sparse. When the Ridge penalty, λ_2 dominates, the model increases in complexity and retains the majority of the features in the model. This process occurs until the mid-2010s, where $\sim 15 - 30$ features are retained, and as many as 50 features are retained during the 2020 Pandemic.

A similiar process occurs with the PCR and PLS models, except there is a more significant amount of variation between the complexity based on the time interval. For the PCR, the model used the maximum number of components during the early 1990s and early 2000s, whereas the PLS used the most components in the early 1980s.

4.3 Model Selection Overview

Now, we can compare the best algorithms among the simple and complex models, which leaves us with the simple linear regression, elastic net, partial least square regression, principal component regression, and the neural network with five hidden layers and neurons. Table (4.5) outlines each of these models with the out-of-sample R^2 computed for our testing interval.

Table 4.5: Performance of the statistical learning portfolios (Complex Models)

	Actual Returns $\bar{r}_p$	Predicted Returns $\hat{\bar{r}}_p$	Volatility σ_r	Sharpe ratio	Out-of-sample R^2
LR-Full	-3.26%	-3.79%	3.21%	-4.09%	5.27%
Elastic Net	-3.23%	-3.81%	2.64%	-5.00%	2.74%
PLS	-3.31%	-3.45%	3.06%	-3.90%	1.82%
PCR	-3.39%	-3.89%	2.70%	-5.00%	6.12%
NN-5	-3.47%	-2.74%	2.37%	-4.00%	2.06%

If the choice of the machine learning portfolio was based solely on predicted returns, then the $NN - 5$ would be the model that generates the better portfolio (although returns are still quite negative). If the same choice was based on the metric of accuracy, then the PLS algorithm generates the best portfolio, as it matches closely to accurate returns. If we made the same choice based on risk or risk-adjusted returns, then the NN-5 would be preferable for investors who were more risk-adverse; however, the PLS generates higher risk-adjusted returns, which is a metric closely monitored by portfolio managers. Finally, we have the matter of predictability, which is where the PCR dominates, followed by the LR-Full.

In choosing the best machine learning portfolio, we feel that it is a goo balance to choose the model with the most accurate performance, with a good risk-adjusted return. The partial least squares is the model that fits this reasonable criteria.

Chapter 5

Conclusion

5.1 Can Statistical Learning Help?

Earlier, we wondered if methods that deliver the most accurate forecast at the individual stock level also provide the best-performing portfolio once we aggregate across stocks. As we can see from the previous section, the answer is not clear. Statistical methods that yield better predictions for individual stock returns, concerning R^2_{oos}, do not necessarily yield better portfolios when utilizing statistical financial metrics. One reason for this is the low signal-to-noise ratio in financial data.

An appropriate signal-to-noise ratio is achievable as alternatives to other features and data arise and machine learning techniques advance. At that time, the amount of data one needs to create accurate machine-learning models will decrease as the potential gain from these models will increase. However, until that time comes, low signal-to-noise ratios will remain an insurmountable challenge, which will make it empirically difficult to determine stock portfolios can be generated based on their statistical performance.

Bibliography

[1] Fama, E. F., & MacBeth, J. D. (1973). Risk, return, and equilibrium: Empirical tests. Journal of political economy, 81(3), 607-636.

[2] Jegadeesh, N. (1991). Seasonality in stock price mean reversion: Evidence from the US and the UK. The Journal of Finance, 46(4), 1427-1444.

[3] Efficiency, S. M. (1993). Returns to Buying Winners and Selling Losers: Implications for. The Journal of Finance, 48(1), 65-91.

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

[5] Lewellen, J. (2014). The cross section of expected stock returns. Forthcoming in Critical Finance Review, Tuck School of Business Working Paper, (2511246).