HEC MONTRÉAL

Comparison of Machine Learning Models for Forecasting Stock Market Returns

by

Kavian Pourrostami

Under the supervision of

Tolga Cenesizoglu

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Applied Financial Economics (M.Sc.)

Mémoire présenté en vue de l'obtention du grade de maîtrise ès sciences en Économie Financière Appliquée

Abstract

In this thesis, we present a comprehensive comparison of different machine learning methods for forecasting stock market returns. Forecasting stock market returns has always been a popular topic among researchers. This study tests a wide range of models, including traditional linear regression, regularization techniques (Lasso, Ridge, and Elastic Net), ensemble methods (Random Forest and XGBoost), and advanced neural networks such as feedforward neural networks, Recurrent Neural Networks (RNN), and Long Short-Term Memory (LSTM) networks for predicting the S&P 500 index.

The empirical results indicate that, while complex machine learning models such as Random Forest and multi-layer neural networks achieve high R^2 values in-sample, regularization methods such as Lasso and Lasso-based combinations outperform the benchmark model in the Diebold-Mariano test and produce more robust out-of-sample results. Highly complex models—including RNNs, LSTM, and XGBoost—tend to overfit the training data: despite their high in-sample R^2, they do not perform as well during testing or in out-of-sample periods.

Résumé

Dans ce mémoire, nous présentons une comparaison approfondie de différentes méthodes d'apprentissage automatique pour la prévision des rendements boursiers. La prévision des rendements boursiers a toujours été un sujet populaire parmi les chercheurs. Cette étude teste un large éventail de modèles, notamment la régression linéaire traditionnelle, les techniques de régularisation (Lasso, Ridge et Elastic Net), les méthodes d'ensemble (Random Forest et XGBoost) ainsi que des réseaux de neurones avancés tels que les réseaux de neurones feedforward, les réseaux de neurones récurrents (RNN) et les réseaux de mémoire à long terme (LSTM) pour la prédiction de l'indice S&P 500.

Les résultats empiriques indiquent que, bien que des modèles d'apprentissage automatique complexes comme Random Forest et les réseaux de neurones à plusieurs couches obtiennent des valeurs élevées de R^2 in-sample, les méthodes de régularisation telles que Lasso et les combinaisons basées sur Lasso surpassent le modèle de référence au test de Diebold-Mariano et produisent des résultats out-of-sample plus robustes. Les modèles très complexes — incluant les RNN, LSTM et XGBoost — ont tendance à surajuster les données d'entraînement : malgré leurs bons résultats in-sample en R^2, leurs performances sont moins satisfaisantes lors des tests ou en période out-of-sample.

Acknowledgment

I would like to express my heartfelt gratitude to my supervisor, Professor Tolga Cenesizoglu, for his continuous support, guidance, and expertise. His insights and encouragement have played a crucial role in the development and progress of this research.

I am deeply thankful to my mother and father, whose unwavering support has always been a source of strength for me—especially during my studies.

I also wish to sincerely thank HEC Montréal for awarding me the generous tuition fee exemption scholarship, which greatly eased the financial burden during my master's program.

I would also like to thank the Fin-ML program for the meaningful funding they provided for my research.

Utilization of Artificial Intelligence Disclosure

I declare that I used ChatGPT (https://chat.openai.com/) during the research and writing of this thesis for the following purposes:

- Checking for typographical errors and correcting them.
- Receiving assistance with Python coding and debugging.
- Assistance with the creation of Figure 4.3.
- Assisting with LaTeX formatting and structuring the document.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Predicting stock market returns has been one of the most widely studied topics in financial economics for decades. A variety of approaches have been applied, ranging from econometric models such as linear and nonlinear regressions to time-series forecasting techniques. In more recent research, many researchers have explored the use of machine learning methods to forecast equity market returns.

Researchers have widely used non-parametric models for forecasting stock market returns. However, these models face challenges due to their reliance on high-dimensional datasets, leading to the so-called "curse of dimensionality," as they consider too many factors. On the other hand, parametric methods are also used, but they tend to be rigid and often result in overfitting (Rossi 2018).

When a model is trained and evaluated on the same period of data, it can lead to misleading conclusions. Although the selected models may perform well in-sample, they often fail to generalize out-of-sample. Several studies, including D. B. Nelson and Kim (1993), Stambaugh (1999), J. Y. . Campbell and Yogo (2006), and Lewellen et al. (2010), have highlighted these issues. While some of these biases can be mitigated using statistical methods, they cannot be completely eliminated (Rossi 2018).

In their influential article, Welch and Goyal (2008) evaluate the performance of several OLS models for stock return prediction by employing measures such as R^2, SSE, and RMSE, and benchmark these models against a simple historical average. Their findings indicate that most OLS models generally fail to outperform the historical average in out-of-sample forecasts. Building on this, subsequent research such as J. Y. Campbell and Thompson (2008) investigates whether introducing certain restrictions—like exclud-

ing predictors with negative coefficients or generating forecasts only when the predicted return is positive—can enhance predictive accuracy. These methodological adjustments aim to align model forecasts more closely with established economic theory. In this thesis, a central question is whether the models we develop can statistically outperform the historical average, and whether R^2 is an appropriate metric for evaluating such outperformance.

In this research, we employed a variety of machine learning models—including regularization techniques, neural networks, random forests, LSTM, and RNN—to predict aggregate stock market returns, measured as monthly S&P 500 index returns.

We began our analysis with linear regression, followed by the application of regularization techniques such as Ridge, Elastic Net, and Lasso regression to determine whether these methods—by reducing the dimensionality of our dataset—could improve the forecasting accuracy of linear models. Next, we incorporated statistical techniques like Principal Component Analysis (PCA) and Partial Least Squares (PLS), combining them with linear regression, Lasso, Ridge, and Elastic Net to assess whether these integrated approaches could further enhance model performance.

In the following stage, we explored more advanced machine learning models. Neural networks, which attempt to mimic how the human brain processes information, and random forest, which splits the data into subgroups and fits a model to each, allow for more flexible pattern detection. We also implemented XGBoost, an ensemble method that combines multiple weak learners to construct a more robust model.

Finally, we examined LSTM and RNN models, which are types of neural networks with recursive architectures. RNNs are designed with memory capabilities, while LSTMs feature both short-term and long-term memory. Our objective was to investigate whether these memory-enhanced models could improve the prediction of stock market returns.

Overall, our goal is to determine which of these machine learning models perform best in forecasting stock market returns and to assess how their predictive accuracy can be effectively measured. To this end, we utilize traditional performance metrics such as R^2 as well as statistical tests like the Diebold-Mariano test to compare model performance.

Our findings show that complex models such as RNNs, LSTMs, and XGBoost, although they perform well in terms of R^2 in the in-sample evaluation, tend to perform poorly out-of-sample, which can be caused by overfitting. On the other hand, simpler

models such as linear regression and regularization methods (Lasso, Ridge, Elastic Net), despite not showing very high in-sample R^2, sometimes outperform more complex models out-of-sample and produce positive R^2 values. Combining these methods with PLS and PCR can improve their forecasting power in certain cases.

Famous models such as Random Forests and multilayer neural networks, although they showed positive out-of-sample R^2, did not demonstrate statistically significant improvement over the naive model (a simple average) according to the Diebold-Mariano test used in our research.

In terms of robustness, we observed that parameters such as the regularization parameter λ and the rolling window size can have a significant effect on our results and on the forecasting accuracy metric R^2 used in this study.

Chapter 2

Literature review

Predicting stock market returns has been a popular topic among researchers for nearly a century, with efforts tracing back to the 1920s. A well-known study by Welch and Goyal (2008) introduced an approach where the equity premium at time t was regressed on various lagged predictor variables. Despite strong theoretical and empirical interest in these predictors, their effectiveness has remained contentious.

In their comprehensive analysis, Welch and Goyal (2008) tested a wide range of predictors for the equity premium that had been previously suggested in the academic literature. Their study examined financial variables such as dividend-price ratios, earnings-price ratios, interest rates, inflation rates, book-to-market ratios, Treasury Bill rates, term spreads, and stock variance to evaluate model performance. The authors divided the data into two main periods: the in-sample (IS) period, where models were trained, and the out-of-sample (OOS) period, where models were tested on previously unseen data. They chose a rolling window of 120 months (10 years) for the in-sample period and used monthly U.S. stock market data from 1926 to 2006. For each step, the models were evaluated out-of-sample using the subsequent month, and the training window was moved forward by one month each time. The OLS model, in particular, performed poorly in terms of R^2, especially during out-of-sample evaluation.

Their findings show that most models struggled to maintain predictive power over time—particularly when more recent data and major economic shocks, such as the Oil Shock of the 1970s (which affected many countries, including the U.S.), were included. This underscores the inherent difficulty of using traditional financial indicators to forecast equity premiums, as well as the complexity of predicting stock market returns with

simple linear models when stock market dynamics are often driven by complex, nonlinear patterns.

Machine Learning in Empirical Asset Pricing

With the introduction of machine learning into empirical asset pricing, new approaches have emerged for forecasting stock returns. The study by Gu et al. (2020) offers a comprehensive analysis of how various machine learning methods perform in this context. While traditional regression-based models remain foundational and provide strong causal interpretation, they often fall short in capturing the complex and nonlinear structures observed in today's financial markets.

Gu et al. (2020) conducted an extensive empirical analysis involving nearly 30,000 individual stocks over a 60-year period (1957–2016). They employed a wide set of predictors and applied advanced machine learning techniques to forecast stock returns. Their results demonstrate that nonlinear models such as neural networks and random forests consistently outperform traditional approaches like OLS. In addition to predictive modeling, they utilized dimensionality reduction techniques such as Principal Component Analysis (PCA), Partial Least Squares (PLS), and regularization methods to identify the most influential variables. These reduced feature sets were then used as inputs to machine learning models, further enhancing predictive performance.

Furthermore, the authors tested various neural network architectures by varying the number of hidden layers. Among the models they evaluated, the three-layer neural network (referred to as NN3) achieved the best out-of-sample R^2, highlighting the advantage of deeper architectures in capturing nonlinear relationships in the data. Their findings indicate that applying machine learning methods can significantly improve forecasting of stock market returns, especially for out-of-sample predictions.

The authors also enhanced the R^2 of linear models by applying Principal Component Regression (PCR). In contrast, machine learning models—such as random forests and neural networks—demonstrated substantial improvements by effectively handling nonlinearity and uncovering hidden patterns in large datasets with numerous predictor variables.

Moreover, Gu et al. (2020) identified consistent predictive signals across various meth-

ods, including momentum, liquidity, and volatility. This consistency suggests that while underlying financial phenomena persist, machine learning models are better equipped to detect and leverage these signals due to their flexibility and robustness. The authors also discussed practical considerations for implementing these models, such as sample splitting for estimation and validation, hyperparameter tuning, and the use of robust objective functions to avoid overfitting and enhance predictive performance.

Regularization Methods

Regularization techniques such as Ridge, Lasso, and Elastic Net have become essential tools in modern statistical learning and are widely used for forecasting financial data. Ridge regularization, introduced for linear models, addresses multicollinearity among predictors by shrinking coefficients and stabilizing model estimates (Hastie 2020). This approach is especially valuable in high-dimensional settings, where the number of predictors may exceed the number of observations. Regularization not only helps prevent overfitting but can also aid in selecting the most relevant variables by reducing the influence of less informative predictors.

Lasso, introduced by Tibshirani (1996), modifies the ordinary least squares (OLS) cost function by adding an ℓ_1 regularization term, which is the sum of the absolute values of the coefficients. This allows Lasso to perform variable selection by shrinking some coefficients exactly to zero, resulting in more interpretable models. The resulting sparsity makes Lasso especially valuable when only a subset of predictors is believed to be relevant.

Ridge, introduced by Hoerl and Kennard (1970), is another regularization method that modifies the OLS cost function by adding an ℓ_2 penalty term, which is proportional to the sum of the squared coefficients. Ridge regularization reduces the size of some coefficients but does not set any of them exactly to zero.

Elastic Net, proposed by Zou and Hastie (2005), combines the strengths of both Ridge and Lasso by including both ℓ_1 and ℓ_2 penalties. This approach is particularly effective when predictors are highly correlated, as it tends to select groups of related variables rather than choosing only one.

Building on these methods, Yuan and Lin (2007) developed a group-based regular-

ization approach that applies Lasso penalties at the group level, allowing for structured variable selection. This method is particularly well-suited for domains where predictors are naturally grouped, such as industry sectors in finance or categories in textual data.

A related study by Neba et al. (2023) conducted a comparative analysis of various regression models, including Ridge, Lasso, Elastic Net, and Random Forest, to predict Netflix's stock prices. In their research, they evaluated model performance using the Root Mean Squared Error (RMSE) as the primary metric. Their findings showed that Lasso achieved the lowest RMSE, closely followed by Elastic Net. Both models outperformed Random Forest, highlighting the effectiveness of regularization techniques in stock price forecasting.

Similarly, Ding (2023) applied the Elastic Net method to forecast the closing price of Apple stock. In their study, various forecasting accuracy metrics—such as Mean Squared Error (MSE) and R-squared—were used to assess the model's performance. The results indicated that Elastic Net effectively captured patterns in Apple's closing price, reinforcing its suitability for financial time series forecasting.

In their research, Chun et al. (2024) used a variety of machine learning models, including penalized linear methods like Lasso and Elastic Net, as well as deep learning and tree-based models, to forecast stock returns and exchange rates. They employed 137 different financial and economic indicators from both the Korean and U.S. stock markets to predict stock market returns. They found that Lasso and Elastic Net outperformed traditional benchmarks in forecasting accuracy.

These regularization approaches have been successfully applied across various areas in economics and finance, offering both improved predictive performance and interpretability. Due to their simplicity and the clarity they provide, these methods form the methodological foundation for many modern machine learning models used in empirical financial forecasting.

Tree-Based and Ensemble Methods

Decision trees, first introduced by Breiman et al. (1984), are among the most popular machine learning methods for classification tasks. Their appeal lies in their intuitive structure and the use of efficient greedy algorithms, which quickly identify key decision

points for classifying data and allow the model to be run separately for each group to find the best fit for each category. However, checking all possible splits is both time-consuming and computationally costly. As a result, decision trees typically select the best split at each step without considering the implications for future splits (Kelly and Xiu 2023).

XGBoost, introduced by Chen and Guestrin (2016), is a scalable machine learning method that employs tree boosting. It builds on the idea of combining multiple weak learners—typically decision trees—in a sequential manner, where each new tree aims to correct the errors of its predecessors. XGBoost minimizes a loss function through additive model updates, making it highly effective at handling complex, nonlinear relationships in data.

One of the standout features of XGBoost is its ability to handle sparse data. It introduces a sparsity-aware split-finding algorithm that enhances computational efficiency by focusing only on non-missing entries, thereby reducing time complexity. Additionally, XGBoost employs a weighted quantile algorithm to find optimal split points in large datasets.

In related research, Zhang (2023) used the XGBoost algorithm to improve time series forecasting in the stock market by predicting patterns in high-frequency time series data. They also applied regularization techniques to help prevent overfitting. Their findings, evaluated using Mean Squared Error (MSE), show that the XGBoost algorithm improves predictive accuracy, particularly in short-term forecasts.

Similarly, Wang (2022) focused on Chinese stock market returns, comparing the performance of three models—XGBoost, Random Forest, and Ordinary Least Squares (OLS)—in forecasting stock returns using financial indicators. Their results indicate that both XGBoost and Random Forest outperformed OLS, with Random Forest slightly outperforming XGBoost. However, the study noted that using only 640 trading days of data to train the models might not be sufficient to fully capture the complexity of the stock market.

Advanced Machine Learning Models: Neural Networks and RNNs

There is a long history of using advanced machine learning methods such as neural networks for financial forecasting. As an early example, White (1988) compared neural network methods with the autoregressive (AR) model for predicting daily IBM stock returns and found that neural networks performed better than the AR model, which served as the benchmark. This conclusion was based on a comparison of the R^2 values for the two models.

In another research, Atsalakis and Valavanis (2009) used artificial neural networks (ANNs), including feedforward, recurrent, and hybrid models, to predict stock prices, returns, and indices for stock market forecasting. Their findings suggest that these models can capture the nonlinear patterns of financial data much better than traditional models such as ARIMA. The study also highlights the importance of training neural networks properly and selecting appropriate input variables when using ANN models.

Recursive neural networks (RNNs) and Long Short-Term Memory (LSTM) networks are widely used in time series analysis and sequence modeling, valued for their ability to capture information from previous time steps that influence future outcomes. However, RNNs often face challenges such as vanishing and exploding gradients. LSTM networks, with their more sophisticated neural architecture and gating mechanisms, address these issues by regulating the flow of information more efficiently. These gates help LSTM models maintain long-term dependencies more effectively (Cong et al. 2020).

D. M. Q. Nelson et al. (2017) used Long Short-Term Memory (LSTM) networks for forecasting stock market returns. They applied the LSTM model to predict Brazilian stock prices using historical data and technical indicators. Specifically, their model was designed to classify whether the stock price would rise in the next 15 minutes, framing the problem as a binary classification task. The predictive accuracy of the LSTM model was evaluated and compared against other benchmark models, such as Multi-Layer Perceptrons (MLP) and Random Forests, using metrics including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and directional accuracy (the proportion of correctly predicted upward and downward movements). Their findings showed that the LSTM model outperformed these traditional models in terms of predictive accuracy.

In another study, Zhu (2020) used Recurrent Neural Network (RNN) models to predict Apple's stock price using data from Yahoo Finance over a ten-year period. They implemented a two-layer RNN architecture and evaluated the model's performance using Mean Squared Error (MSE) and Mean Absolute Error (MAE) metrics. The study found that RNNs are effective for predicting short-term movements in stock market returns. The close match between the predicted and actual values is shown in Figure 2.1, where the RNN model was applied with a timestep of 10.

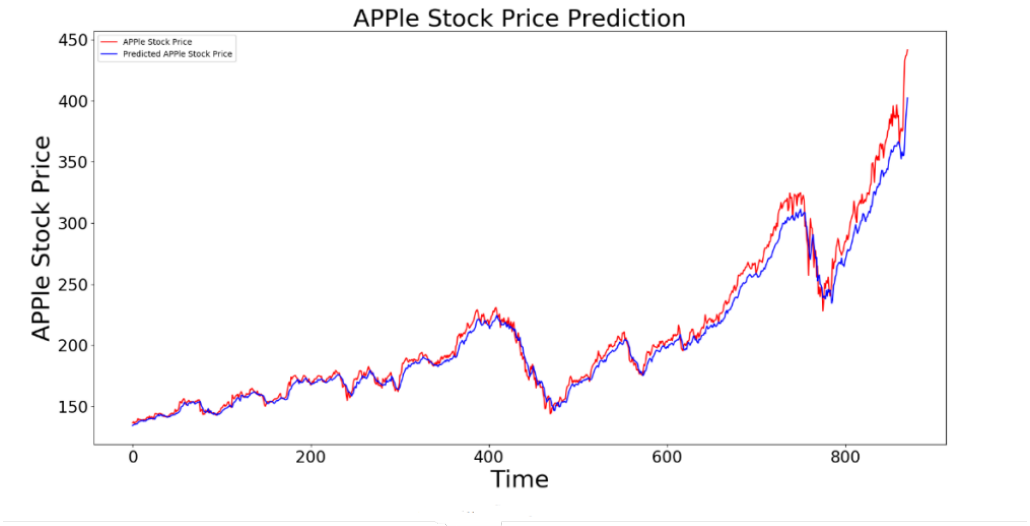

Figure 2.1: Predicted vs. actual Apple stock price using RNN model with timestep = 10 (Zhu 2020).

In financial markets, LSTM models have also been applied to historical asset data, enhancing predictions of asset returns and their distributions (Cong et al. 2020). Despite their advantages, LSTMs are not without limitations; they can still experience gradient-related issues and instability in input feature usage, sometimes leading to extreme outputs. Thus, advanced RNNs may not always be the ideal choice for every application.

Chapter 3

Data

3.1 Dataset

In this analysis, we use monthly S&P 500 data from January 1871 to December 2021. The dataset comprises 21 key variables essential for financial analysis. These include the market index value (Index), dividend yield (D12), earnings yield (E12), and the book-to-market ratio (b/m). Interest rates are represented by the Treasury bill rate (tbl), yields on AAA-rated and BAA-rated corporate bonds (AAA and BAA), and the long-term government bond yield (lty).

Other variables include net total issuance (ntis), long-term returns (ltr), and corporate bond returns (corpr). Market volatility is captured by stock variance (svar), while the dataset also includes value-weighted returns on the CRSP index, both with and without dividends (CRSP_SPvw and CRSP_SPvwx). Additionally, the dataset features the dividend-price ratio (dp), earnings-price ratio (ep), dividend-earnings ratio (de), term spread (tms), default yield spread (dfy), risk-free rate (rf), and the market return minus the risk-free rate (rmrf).

These variables provide a comprehensive foundation for forecasting the S&P 500 using different methods. In my analysis, I calculated several financial factors using the initial variables in the dataframe. Specifically, I computed the dividend ratio and earnings ratio by dividing the past 12 months' dividends (D12) and earnings (E12) by the index, respectively. Additionally, I calculated the dividend-earnings ratio by dividing the past 12 months' dividends (D12) by earnings (E12) to assess how much of the earnings have been paid out as dividends to shareholders. I also calculated the term spread (tms) by

subtracting the Treasury bill rate from the long-term government bond rate, which illustrates the difference between long-term and short-term interest rates—a useful indicator of economic conditions. Lastly, I derived the default yield spread (dfy) by subtracting the AAA corporate bond rate from the BAA rate, providing a measure of credit risk. These parameters help improve the accuracy of stock market return predictions and are valuable tools for financial analysis.

As shown in Table 3.1, stock market returns have changed over time. To understand these movements, we examined the summary statistics for our main target variable, `CRSP_SPvw`, covering the period from June 1947 to November 2022. During this time, the average monthly return across all 10-year subperiods remained positive, generally ranging between 0.5% and 1.5%. The standard deviation of returns—our measure of volatility—was relatively stable, mostly falling between 3.4% and 4.5%. However, in the most recent period (2017–2022), volatility increased slightly to 5.12%. It is important to note that this latest subperiod covers only 5 years and 5 months, not a full decade, which could partially explain the higher volatility. Still, this may suggest that the market has become more volatile in recent years.

The largest monthly decline recorded was approximately –21.6%, while the highest gain reached about 16.8%. Although maximum and minimum returns varied across different decades, the fact that both the average and median returns remained positive underscores the market's long-term growth. These summary statistics help us understand the overall behavior of the data before applying any forecasting models.

Period	Count	Mean	Std Dev	Min	25%	50%	75%	Max
1947-06-01 to 1957-06-01	121	0.0153	0.0372	-0.0983	-0.0133	0.0170	0.0459	0.0958
1957-06-02 to 1967-06-02	120	0.0088	0.0339	-0.0807	-0.0112	0.0150	0.0307	0.1083
1967-06-03 to 1977-06-03	120	0.0048	0.0453	-0.1175	-0.0206	0.0022	0.0375	0.1681
1977-06-04 to 1987-06-04	120	0.0142	0.0434	-0.0975	-0.0132	0.0139	0.0419	0.1352
1987-06-05 to 1997-06-05	120	0.0123	0.0409	-0.2158	-0.0070	0.0157	0.0379	0.1141
1997-06-06 to 2007-06-06	120	0.0069	0.0434	-0.1431	-0.0175	0.0109	0.0374	0.0985
2007-06-07 to 2017-06-07	120	0.0068	0.0436	-0.1670	-0.0151	0.0123	0.0333	0.1090
2017-06-08 to 2022-11-01	65	0.0109	0.0512	-0.1220	-0.0161	0.0203	0.0406	0.1289

Table 3.1: Summary Statistics of CRSP_SPvw Returns by Decade (1947–2022)

3.2 Data processing

The first step in processing our data is to assess the relevance of the features, identifying which ones are most useful for predicting the target variable. This helps reduce the complexity of the problem. The Pearson correlation coefficient (PCC) is a classical method for measuring the linear correlation between two variables (Niu et al. 2022). PCC values range from –1 to 1, with higher absolute values indicating stronger linear relationships between features and the target variable. In our analysis, we used this approach to identify and visualize the most influential predictors.

Figure 6 presents the Pearson correlation coefficient (PCC) chart for our data. This chart highlights the correlations between different features, allowing us to assess the relevance and relationships among them.

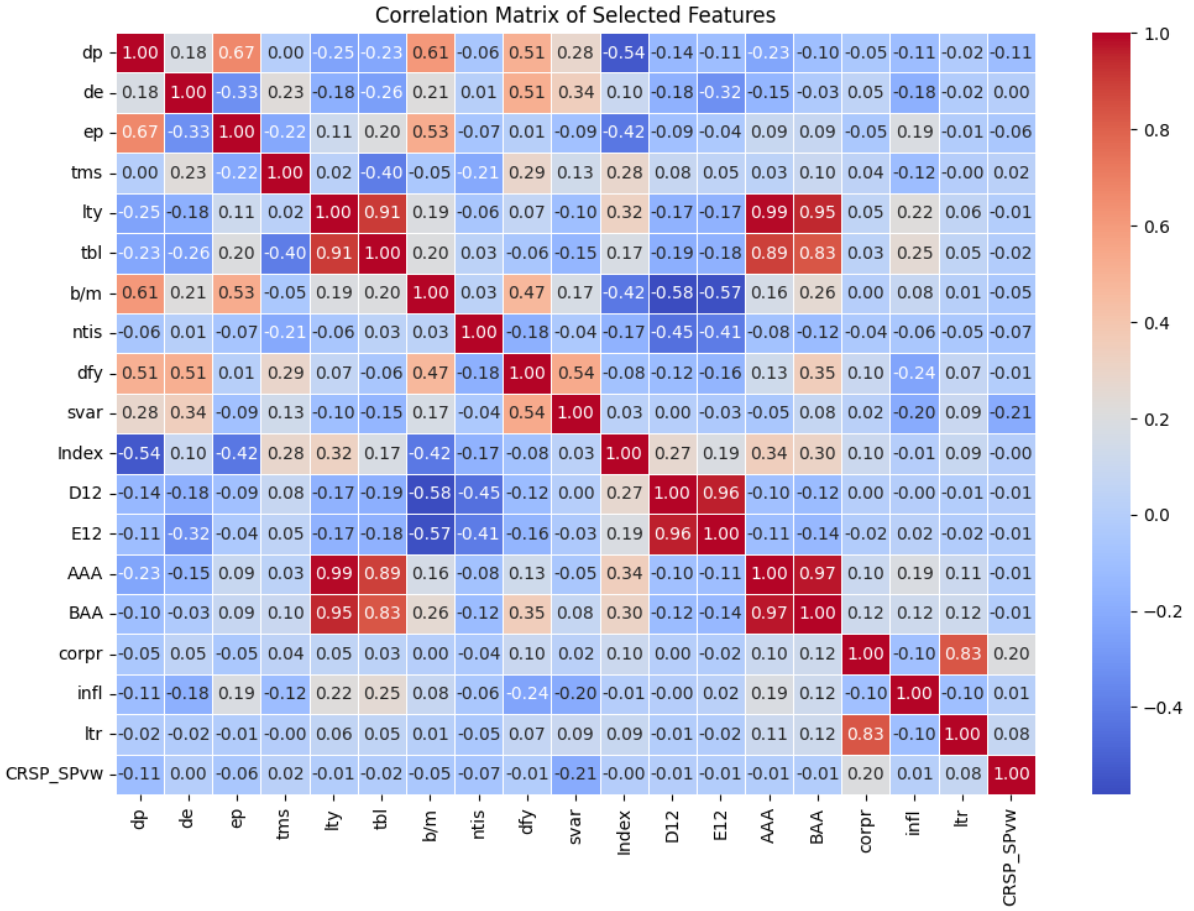

Figure 3.1: Correlation Matrix of Selected Features

Chapter 4

Mthodology

4.1 Linear Regression

Linear regression models the relationship between a dependent variable y and one or more independent variables X. The simple linear regression model can be represented as:

$$y = X\beta + \epsilon \tag{4.1}$$

In Equation 4.1, y is the vector of observations, X is the matrix of input features, β is the vector of coefficients, and ϵ is the error term. The coefficients β, as shown in Equation 4.2, are typically estimated using the Ordinary Least Squares (OLS) method, which minimizes the residual sum of squares:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - X_i\beta)^2 \tag{4.2}$$

Linear regression is considered one of the simplest machine learning methods. In our experiment, we used a rolling window approach with a size of 120 months (10 years) to train and test the model. We advanced the window one month at a time, regressing the value-weighted return of the stock market on various features. The results were not promising, especially for the out-of-sample R^2 values.

We also tested model performance using the top three principal components from Principal Component Analysis (PCA) and the top three features from Partial Least Squares (PLS) as inputs, to assess whether dimensionality reduction could further enhance the predictive power of linear regression. Notably, PCA selects components with the greatest

variance-explaining power, while PLS identifies features most correlated with the target variable. As a result, these approaches can improve the interpretability of our regression models.

4.2 Lasso

Lasso (Least Absolute Shrinkage and Selection Operator) adds an L_1 regularization term to linear regression, as shown in Equation 4.3, allowing it to shrink some coefficients to zero. This effectively selects a simpler model and reduces the dimensionality of the data:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left(\sum_{i=1}^{n} (y_i - X_i\beta)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right) \tag{4.3}$$

Unlike Ridge regression, Lasso performs both shrinkage and variable selection by forcing some coefficients to be exactly zero. This makes it especially useful when we suspect that only a few predictors are truly relevant.

In our experiment, we set $\lambda = 0.0001$, as higher values could shrink all coefficients to zero. We applied the same rolling window strategy as in linear regression, using a 120-month window to train the model and predicting the 121^{st} observation. This process was repeated by sliding the window forward one month each time the model was run.

We also tested model performance using the top three principal components from PCA and the top three features from PLS as inputs, to assess whether dimensionality reduction could further enhance the predictive power of Lasso regression. Notably, PCA selects components with the greatest variance-explaining power, while PLS identifies features most correlated with the target variable. As a result, these approaches can improve the interpretability of our Lasso regression models. Lasso regression is similar to linear regression, but it incorporates a shrinkage (regularization) term that penalizes large coefficients.

4.3 Ridge

Ridge regression, also known as L_2 regularization, addresses the issue of multicollinearity by modifying the least squares objective function. It adds a penalty to the size of the coefficients to prevent overfitting, as shown in Equation 4.4:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left(\sum_{i=1}^{n} (y_i - X_i\beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right) \tag{4.4}$$

This technique is especially helpful when predictor variables are highly correlated, which can cause the design matrix $X^T X$ to become nearly singular. By adding λI to the diagonal, Ridge regression ensures that all eigenvalues are positive, thereby improving numerical stability.

Ridge regression reduces the variance of the estimates at the cost of introducing a small amount of bias. This trade-off often leads to better generalization in high-dimensional settings.

In our experiment, we set $\lambda < 0.001$ to avoid excessive shrinkage. We used the same 10-year rolling window approach as in previous models, aiming to improve prediction accuracy by stabilizing the coefficient estimates rather than eliminating them.

As with linear regression and Lasso regression, we also trained our Ridge model using the top three features selected by both PCA and PLS to examine whether this approach could further improve model performance.

4.4 Elastic Net

Elastic Net combines both L_1 (Lasso) and L_2 (Ridge) regularization, providing a balanced approach that leverages the strengths of both techniques. This method is particularly advantageous when dealing with datasets that exhibit high multicollinearity or when feature selection is important. The objective function of Elastic Net is given in Equation 4.5:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left(\sum_{i=1}^{n} (y_i - X_i\beta)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \right) \tag{4.5}$$

In Equation 4.5, λ_1 and λ_2 are the regularization parameters associated with the Lasso and Ridge penalties, respectively. In practice, these terms are often combined through a mixing parameter α, such that:

$$\lambda_1 = \alpha\lambda, \quad \lambda_2 = (1 - \alpha)\lambda$$

where λ is the overall regularization strength and $\alpha \in [0, 1]$ determines the balance between the Lasso (L_1) and Ridge (L_2) components. In our study, we set $\alpha = 0.5$, giving

equal weight to both penalties. This approach allows us to benefit from the sparsity induced by Lasso and the stability provided by Ridge regularization.

We applied the same rolling window procedure as described earlier, using 120 months of data for training and generating out-of-sample predictions for the subsequent month. In each window, the Elastic Net regression leveraged the strengths of both Lasso and Ridge penalties through the specified mixing parameter, as described above.

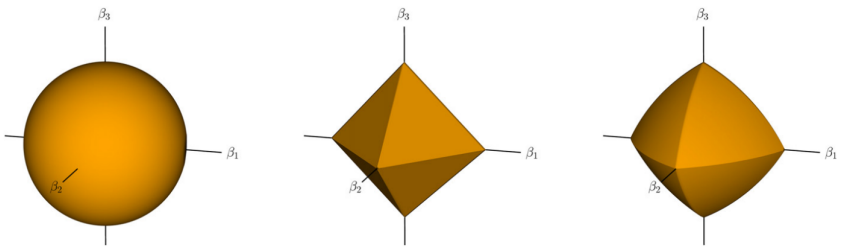

Figure 4.1: Constraint balls for Ridge, Lasso, and Elastic Net regularization. The sharp edges and corners of Lasso and Elastic Net allow for variable selection in addition to shrinkage (Hastie 2020).

Figure 4.1 illustrates the geometric constraints imposed by each regularization method. While Ridge's circular constraint leads to uniform shrinkage, the diamond shapes of Lasso and Elastic Net introduce edges that can push some coefficients to exactly zero, thereby enabling feature selection as well as shrinkage.

4.5 Neural Networks

Neural networks are a class of models inspired by the structure and function of the human brain. They consist of layers of interconnected nodes, commonly referred to as neurons. The output of a neural network with L layers can be expressed as shown in Equation 4.6:

$$\hat{y} = f\left(W^{[L]} \cdot f\left(W^{[L-1]} \cdot \ldots f\left(W^{[1]} \cdot X + b^{[1]}\right) + \ldots + b^{[L-1]}\right) + b^{[L]}\right) \tag{4.6}$$

In Equation 4.6, $W^{[l]}$ and $b^{[l]}$ denote the weights and biases for the l^{th} layer, respectively, and f represents the activation function.

Neural networks aim to mimic certain processes of the human brain and are widely used in various applications, particularly in data science, such as prediction, computer

vision, and classification tasks. These models are composed of multiple layers, each containing several neurons (nodes). In each layer, every input is multiplied by a corresponding weight, and the results are summed and passed through an activation function to produce the output of each neuron, as shown in Equation 4.7:

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{4.7}$$

The concept of neural networks was first introduced in 1943 by McCulloch and Pitts (McCulloch and Pitts 1943). In the earliest models, a threshold was used to compare the output of the neuron's function in order to determine whether it should "fire" (i.e., transmit the signal to the next neuron) or remain inactive. In modern neural networks, this threshold mechanism is typically replaced by a bias term, which serves a similar purpose but offers greater flexibility during training.

There are several types of neural networks. Some, such as feedforward models, pass information only in one direction—from input to output. Others, such as recurrent neural networks, include feedback loops that allow information to flow backward between neurons. The primary goal of all these models is to discover patterns in data. In this thesis, I apply various neural network architectures to forecast stock market returns using my dataset.

4.5.1 Activation Function

Activation functions are a crucial component of neural networks. Without them, neural networks would be equivalent to linear models, lacking the ability to capture complex nonlinear patterns in data.

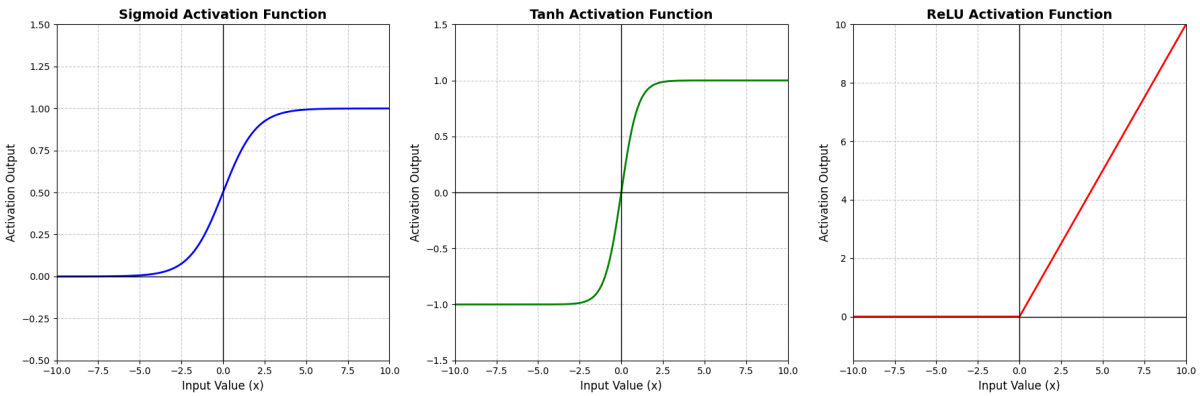

Figure 4.2: Comparison of activation functions: Sigmoid, Tanh, and ReLU.

As illustrated in Figure 4.2, the three most common activation functions are Sigmoid, Tanh, and ReLU. Since our goal is to predict the returns of the stock market - which can be both positive and negative - we selected the Tanh activation function for our models.

.

4.5.2 Neural Architecture

We experimented with different neural network architectures, ranging from a simple network (NN1) with one hidden layer to more complex networks (NN2 to NN5) with up to five hidden layers. The number of hidden layers can significantly influence the model's performance, as shown in Figure 4.3. As the number of hidden layers increases in our models, the number of neurons also rises, resulting in more calculations per epoch. Each epoch corresponds to one complete pass of the data through all the layers. This increase in complexity can yield a more powerful model with greater forecasting capacity.

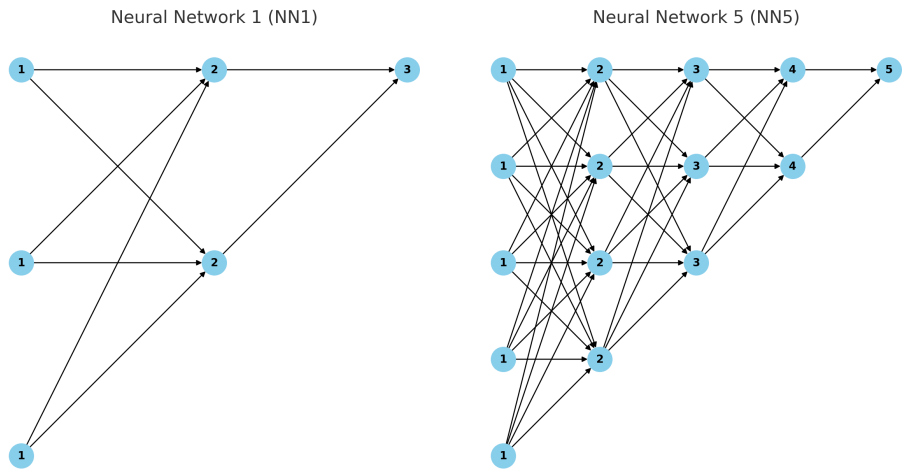

Figure 4.3: Neural Network Architectures: NN1 and NN5

4.5.3 Cross-Validation with Neural Networks

To further refine our approach, we integrated cross-validation with our neural network models. We tested five different neural architectures—two NN4 models and three NN5 models—across each in-sample period, with each period spanning ten years (120 months). For each period, the model with the highest R^2 value was selected for out-of-sample prediction. The results showed a slight improvement in performance.

4.6 Random Forest

Decision trees are among the most common regression methods used for forecasting. They work by splitting the data based on different features at each step and performing a regression within each split. Decision trees are also well known as powerful clustering algorithms. Random Forest is an ensemble learning method that improves upon decision trees by building multiple trees and combining their outputs to enhance accuracy and reduce overfitting. The Random Forest model is represented by Equation 4.8:

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} \hat{y}_b(x) \tag{4.8}$$

In the Equation 4.8, $\hat{y}_b(x)$ is the prediction from the b^{th} tree, and B is the total number of trees in the forest. One of the key parameters in a Random Forest model is the number of estimators, which refers to the number of decision trees used in the ensemble. In our

study, we implemented the Random Forest algorithm with varying numbers of decision trees and evaluated its performance. As in previous sections, we used an in-sample period of 120 months and an out-of-sample period of one month, moving forward month by month.

4.7 XGBoost

XGBoost (Extreme Gradient Boosting) is a highly efficient and scalable machine learning algorithm designed for supervised learning tasks. It is based on the concept of gradient boosting, in which a number of weak learners—typically shallow decision trees—are combined. These trees are trained sequentially, with each new tree attempting to correct the errors made by the previous ones. This iterative process helps build a stronger predictive model over time.The objective function for XGBoost is shown in Equation 4.9:

$$\text{Obj}(\theta) = \sum_{i=1}^{n} l(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{4.9}$$

In the Equation 4.9, $l(\hat{y}_i, y_i)$ represents the loss function, which measures how well the model fits the training data. The term $\Omega(f_k)$ is a regularization component that penalizes model complexity and helps prevent overfitting. $\hat{y}_i$ is the prediction for the i^{th} instance.

What distinguishes XGBoost from traditional boosting algorithms is its use of both the first and second derivatives (the gradient and the Hessian) of the loss function during optimization. This feature enables faster and more accurate learning. XGBoost also supports parallel computing, dividing the overall task into smaller subtasks that are processed simultaneously, which makes it especially effective for large-scale, high-dimensional datasets. Additionally, it natively handles missing data, contributing to its robustness in practical applications.

XGBoost further enhances the gradient boosting framework by incorporating a regularized learning objective that penalizes model complexity. This helps address overfitting and improves generalization. The regularized objective function is defined in Equation 4.10:

$$L(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k), \tag{4.10}$$

where $\hat{y}_i$ is the predicted value for the i-th instance, l is a differentiable convex loss function, and $\Omega(f_k)$ is the regularization term that penalizes the complexity of the k-th tree f_k. The regularization term is given by Equation 4.11:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2, \tag{4.11}$$

where T is the number of leaves in the tree, w represents the weights on the leaves, and γ and λ are regularization parameters. This framework encourages the selection of simpler models, reducing the risk of overfitting and improving the model's ability to generalize to unseen data.

In simpler terms, XGBoost builds an ensemble of small decision trees, each improving upon the previous one, to produce a highly accurate final model.

In our experiment, we implemented XGBoost using 5,000 estimators and applied it using the same 10-year rolling window structure as our other models. We evaluated its performance using out-of-sample predictions, aiming to compare its accuracy with both traditional linear models and other machine learning approaches.

4.8 Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) are a type of neural network specifically designed to handle sequential or time-based data. Unlike traditional feed-forward neural networks, which treat each input as independent from the others, RNNs are built to retain information over time. They achieve this by passing information from one step of the sequence to the next, allowing past inputs to influence future predictions.

This memory mechanism makes RNNs particularly useful for tasks where the order and context of the data matter—such as speech recognition, language modeling, and, in our case, predicting stock market returns over time.

What makes RNNs unique is their looped architecture, which maintains a hidden state that carries information forward across time steps. This structure enables them to learn temporal patterns in data. However, standard RNNs often face challenges when attempting to capture long-term dependencies due to issues such as the vanishing gradient problem.

In financial modeling, RNNs are a natural fit because stock market data unfolds over

time and is influenced by past events. Whether it's economic indicators or previous price movements, historical context matters. RNNs are well-suited to capture this kind of structure, making them a powerful tool for time-series forecasting. The basic structure and unrolling mechanism of an RNN is illustrated in Figure 4.4.

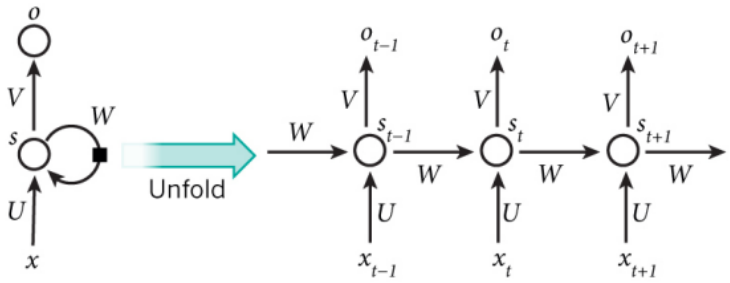

Figure 4.4: Unrolled structure of a standard Recurrent Neural Network (RNN).(Zhu 2020)

4.8.1 Model Architecture

In our experiment, we implemented a basic RNN with the following configuration:

- Input size: 16 (number of features)
- Hidden layer size: 20
- Number of layers: 2
- Dropout: 0.2 (to reduce overfitting)
- Learning rate: 0.001

The RNN was trained using the Adam optimizer, and the loss was measured using Mean Squared Error (MSE), a standard evaluation metric for regression tasks. The model was trained for 100 epochs. We used 75% of the data for training and 25% for testing, allowing us to assess how well the model generalizes to unseen data.

4.9 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are an advanced form of Recurrent Neural Networks (RNNs) designed to capture both short-term and long-term dependencies

in sequential data. While traditional RNNs are effective at learning from recent past information, they often struggle with long sequences due to the vanishing gradient problem—where gradients shrink during backpropagation, making it difficult for the model to learn long-range relationships.

While standard RNNs often face challenges when attempting to capture long-term dependencies due to issues such as the vanishing gradient problem, advanced variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks have been developed to address these issues. LSTMs overcome this limitation by introducing a special architecture that includes memory cells, as well as gates (input, forget, and output gates) that regulate the flow of information. These mechanisms allow LSTMs to selectively retain or discard information over time, enabling them to learn patterns across much longer sequences. As a result, LSTMs are widely used in fields such as speech recognition, text generation, and time-series forecasting—making them a suitable choice for modeling stock return data, which is highly sequential and influenced by both recent and older events. The architecture of the LSTM network used in their work is illustrated in Figure 4.5.

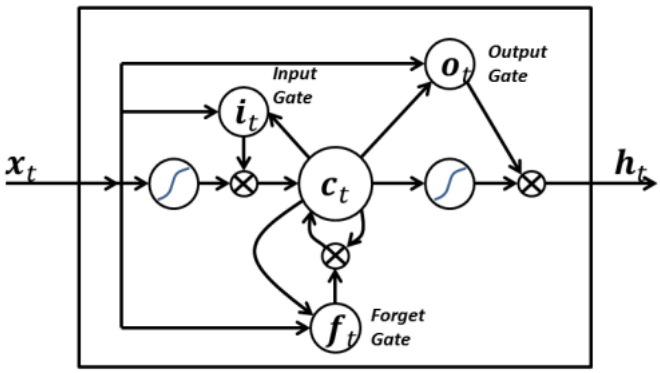

Figure 4.5: LSTM (Long Short-Term Memory) model architecture.(D. M. Q. Nelson et al. 2017)

4.9.1 Model Architecture

For our experiment, we implemented an LSTM model with the following configuration:

- Input Size: 16 (number of input features)

- Hidden Layer Size: 20
- Number of Layers: 2
- Dropout: 0.2 (to reduce overfitting)
- Learning Rate: 0.001

The model was trained using the Adam optimizer, and performance was measured using the Mean Squared Error (MSE) loss function, which is commonly used for regression tasks. Training was conducted over 100 epochs. We used 75% of the data for training and the remaining 25% for testing, allowing us to evaluate how well the model generalizes to unseen data.

By applying LSTM to our stock return forecasting problem, we aimed to leverage its memory capabilities to better capture temporal relationships that may not be immediately obvious from short-term data alone. .

4.10 Performance Evalution

To compare the performance of our models, we first calculate the in-sample and out-of-sample R^2 values using the formula below:

$$R^2 = 1 - \frac{\sum_{t=1}^{n} (y_t - \hat{y}_t)^2}{\sum_{t=1}^{n} (y_t - \bar{y})^2} \tag{4.12}$$

where y_t represents the actual values, $\hat{y}_t$ represents the predicted values, and $\bar{y}$ is the mean of the actual values.

For the models with out-of-sample R^2 scores close to zero (ranging from -20% to 5%), we applied the Diebold-Mariano test (Diebold and Mariano 1995) to assess whether these models performed better than the simple average. The Diebold-Mariano test statistic is calculated as shown in Equation 4.13:

$$DM = \frac{\bar{d}}{\sqrt{\frac{1}{T}\hat{\gamma}_d(0) + 2\sum_{k=1}^{h-1}\hat{\gamma}_d(k)}} \tag{4.13}$$

where:

- $\bar{d} = \frac{1}{T}\sum_{t=1}^{T} d_t$ is the mean loss differential,
- $d_t = L(e_{1,t}) - L(e_{2,t})$ is the difference between the loss functions of two models,
- $\hat{\gamma}_d(k)$ is the autocovariance of d_t at lag k,
- T is the sample size, and
- h is the forecast horizon.

We used the Diebold-Mariano statistical test to evaluate whether each model's out-of-sample performance was better than the simple average of stock market returns within each sample period.

4.11 Applying PCR and PLS for Model Optimization

We used Principal Component Analysis (PCA) and Partial Least Squares (PLS) for dimensionality reduction by applying them to our large dataset before feeding the inputs into our models.

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms a large set of possibly correlated variables into a smaller set of uncorrelated variables, called principal components. These components are ranked by how much of the variance in the original dataset they explain, with the first component capturing the most variance. In practice, PCA helps reduce noise, avoid multicollinearity, and improve computational efficiency by selecting the most informative features. This is especially useful in stock market return forecasting, where identifying key patterns in noisy data can lead to better predictive performance.

Partial Least Squares (PLS) is another dimensionality reduction method; however, unlike PCA, it focuses on the relationship between the input variables and the target variable. PLS identifies new components by maximizing the covariance between the predictors and the response variable. This makes PLS particularly effective for regression tasks, as it reduces data complexity while emphasizing the features most relevant to

prediction. In fields such as finance, where variables may be noisy or only weakly related to the outcome, PLS enhances model interpretability and forecasting accuracy.

First, we applied PCA and selected the top three components based on the amount of variance they explained in the input data. These components were then used to train our models. This approach, known as Principal Component Regression (PCR), has been applied in previous studies where reduced feature sets were used in regression frameworks (Gu et al. 2020).

We then followed the same approach with PLS, this time selecting the top three components based on their correlation with the response variable. These components were also used to train our models, with the goal of improving predictive accuracy by focusing on the most relevant input features.

Chapter 5

Empirical Results

5.1 In-Sample Empirical Results

5.1.1 Linear regression results

In this section, we analyze the R^2 results of our in-sample linear regression approach, as shown in Table 5.1. Linear regression achieves strong performance during the in-sample period, with an R^2 of 44.73%. However, combining linear regression with PCA or PLS does not improve forecasting accuracy in terms of R^2. This result is reasonable because dimensionality reduction techniques such as PCA and PLS reduce the number of predictor variables, and in the in-sample period where the model is trained, having fewer predictors typically leads to a lower R^2.

Model	**In-Sample** R^2
Linear Regression	0.4473
Linear Regression - PCR	0.0721
Linear Regression - PLS	0.1809

Table 5.1: In-Sample R^2 for linear regression models.

5.1.2 Regularization methods

The following analysis examines the R^2 results of our three in-sample regularization methods. As shown in Table 5.2, Ridge regression demonstrates the best performance among them. For all methods, we observe that combining regularization techniques with PCR or PLS leads to a decrease in R^2. This outcome is expected, as both PCR and PLS

are dimensionality reduction techniques, while regularization methods such as Lasso, Ridge, and Elastic Net also reduce the effective dimensionality of the data. Although positive R^2 values remain, substantial reductions in dimensionality result in fewer features in our regression models, which in turn lowers the overall R^2.

Model	**In-Sample R^2**
Lasso Regression	0.2485
Lasso Regression - PCR	0.0114
Lasso Regression - PLS	0.1204
Ridge Regression	0.4712
Ridge Regression - PCR	0.0883
Ridge Regression - PLS	0.2229
Elastic Net Regression	0.3376
Elastic Net Regression - PCR	0.0094
Elastic Net Regression - PLS	0.0899

Table 5.2: Regularization methods: Lasso, Ridge, and Elastic Net with in-sample R^2 values.

5.1.3 Random Forest

In this analysis, we tested different numbers of estimators, each corresponding to a different number of decision trees. The results are shown in Table 5.3. As we can see, the number of estimators does not have a significant effect on the accuracy of our forecasts during the in-sample period. This outcome could be due to overfitting in the random forest models, which may explain why increasing the number of estimators does not lead to a higher in-sample R^2.

Model	**In-Sample R-squared**
100 Estimators	0.8606
1000 Estimators	0.8666
5000 Estimators	0.8649

Table 5.3: In-Sample R^2 for Random Forest Models with Different Numbers of Estimators

5.1.4 XGboost

We applied XGBoost, which leverages a large number of weak learners to produce a stronger forecaster. As shown in Table 5.4, our XGBoost model with 500 estimators performs exceptionally well during the in-sample period, achieving an R^2 of 0.6886. However,

this high R^2 is likely due to overfitting during the training phase, as the model benefits from a large number of learners. As a result, it may not perform as well when tested on data from a different period than the one it was trained on.

Model	**In-Sample** R^2
500 Estimators	0.6886

Table 5.4: In-sample R^2 for XGBoost with 5000 estimators.

5.1.5 Neural networks

In our analysis, we used NN1 to NN5 neural architectures to examine how increasing the number of layers and neurons affects the accuracy of our model's predictions. Specifically, we aimed to determine whether a more complex neural architecture would enhance the model's forecasting power. As shown in Table 5.5, more complex neural architectures do not improve forecasting accuracy during the in-sample period. In fact, simpler architectures such as NN1 and NN2 perform better than NN4 and NN5, which have more layers. In terms of R^2, the simpler neural networks deliver better performance in the in-sample period.

Model	**In-Sample** R^2
NN1	0.2395
NN2	0.2382
NN3	0.2220
NN4	0.1987
NN5	0.1845

Table 5.5: In-sample R^2 for neural network models.

5.1.6 Cross validation and Neural Networks

We further tested five different neural architectures (two NN4 models and three NN5 models) across each of our in-sample periods. In each rolling window, we selected the model with the highest R^2. The results are presented in Table 5.6. As shown, this approach does not outperform the simple NN1 neural network model.

Model	**In-Sample R^2**
Through 5 Neural Architectures	0.2067

Table 5.6: In-sample R^2 for the model selected from five neural architectures.

5.1.7 RNN and LSTM

RNN and LSTM are deep learning methods that leverage long-term memory. To evaluate their performance, we trained both models using 75% of the data. The resulting training set R^2 values are presented in Table 5.7. As shown, both models perform well during the training period.

Metric	**Value**
RNN Train R^2	0.7736
LSTM Train R^2	0.8836

Table 5.7: Training set R^2 values for RNN and LSTM models.

5.2 Out-of-Sample Empirical Results

5.2.1 Linear regression results

We examine the out-of-sample R^2 results for our linear regression model, as shown in Table 5.8. The linear regression model performs poorly during the out-of-sample period, with an R^2 of -0.2406.

To improve performance, we applied Principal Component Regression (PCR) and Partial Least Squares (PLS). In the PCR approach, we selected the top three principal components that captured the highest variance among the predictor variables. In contrast, PLS was used to extract three latent factors that maximized covariance with the target variable, aiming to improve predictive accuracy. These regularization techniques improved our out-of-sample R^2: with PCR, the R^2 increased to -0.0742, and with PLS, it increased to -0.0416. These results suggest that linear regression benefits from these two dimensionality reduction techniques.

As shown in the last column of Table 5.8, the Diebold-Mariano test p-values for the linear regression and PCR-enhanced linear regression models are below 5%. This indicates that, at the 5% significance level, these models outperform the naive benchmark, which is the simple average model. However, the Diebold-Mariano test p-value for the Linear Regression - PLS model is above 5%, meaning that, at this significance level, it does not significantly outperform the simple average model.

The research by Gu et al. (2020) reported positive R^2 values of 0.27% for PLS and 0.26% for PCR, which are notably higher than the values we obtained in our model, namely -7.42% for PCR and -4.16% for PLS.

Model	**Out-of-Sample R^2**	**DM Test Statistic**	**DM Test P-value**
Linear Regression	-0.2406	2.1746	0.0297
Linear Regression - PCR	-0.0742	2.2704	0.0232
Linear Regression - PLS	-0.0416	0.3597	0.7191

Table 5.8: Out-of-sample R^2 and Diebold-Mariano (DM) test statistics for linear regression, PCR, and PLS models.

5.2.2 Regularization methods

We analyze the R^2 results of three different out-of-sample regularization methods. As shown in Table 5.9, the Lasso regularization method demonstrates the best performance

among them. The PCR and PLS techniques improve the prediction accuracy for both Lasso and Ridge regression. As in previous sections, we used the top three components for PCR, which help explain the variance in the predictor variables, and incorporated the PLS method to account for collinearity with the target variable. However, the overall impact of these techniques remains relatively small. Additionally, these dimensionality reduction methods do not appear to influence the performance of the Elastic Net regularization method.

As shown in the last column of Table 5.9, the only model with a Diebold-Mariano test p-value below 5% is Lasso Regression - PLS. This indicates that, at the 5% significance level, we can reject the null hypothesis that the naive model (a simple average) is better than our model. Consequently, we can conclude that Lasso Regression - PLS outperforms the simple average model. However, we cannot make the same claim for the other models based on the Diebold-Mariano test results. This outcome is reasonable, as Lasso Regression - PLS also has the highest R^2 value among the models in Table 5.9.

The results in Table 5.9 show that, in terms of R^2, techniques such as Lasso and Ridge improve predictive performance, particularly Lasso. However, combining these techniques with PCA or PLS does not necessarily lead to further improvements in R^2, suggesting that combining multiple regularization methods may not be an effective strategy for enhancing forecasting accuracy.

Model	**Out-of-Sample R^2**	**DM Test Statistic**	**DM Test P-value**
Lasso Regression	0.0246	1.5067	0.1320
Lasso Regression - PCR	-0.010	1.1849	0.2360
Lasso Regression - PLS	0.064	-3.2761	0.0011
Ridge Regression	-0.1689	1.2246	0.2207
Ridge Regression - PCR	-0.0356	0.9203	0.3574
Ridge Regression - PLS	-0.0625	0.4175	0.6763
Elastic Net Regression	0.0192	-0.7750	0.4383
Elastic Net Regression - PCR	-0.011	0.3362	0.7368
Elastic Net Regression - PLS	-0.0238	0.5736	0.5662

Table 5.9: Regularization methods: Lasso, Ridge, and Elastic Net with out-of-sample R^2 values.

The performance of our Lasso model, which demonstrates the best results among regularization methods over the overall out-of-sample period, is evaluated in this analysis. The R^2 of our predictions is calculated using rolling 10-year windows, with the analysis updated each month. As shown in Figure 5.1, the model's accuracy fluctuates significantly over time, performing worst during the 2008 financial crisis and the surrounding period.

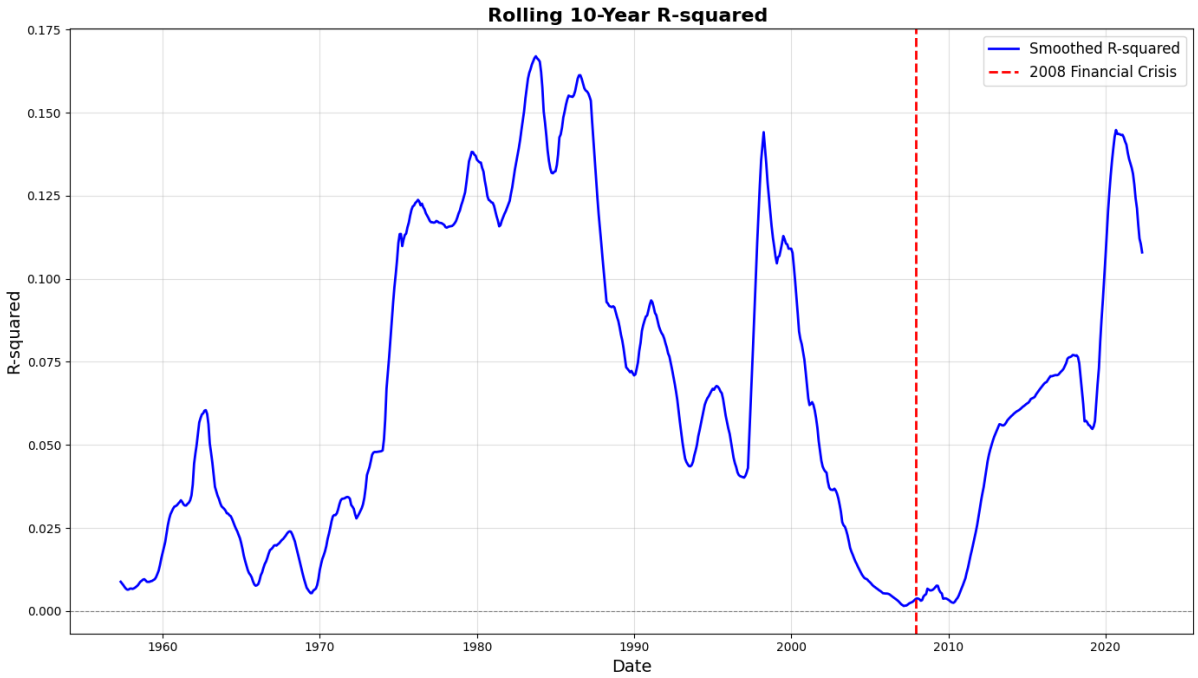

Figure 5.1: Rolling 10-Year R-squared using the Lasso model, with a vertical line marking the 2008 Financial Crisis.

This indicates that the model is effective under stable conditions but struggles during periods of market instability. Notably, Figure 5.1 shows that the R^2 of our predictions across different rolling periods ranges from 0 to nearly 17% for the Lasso model.

5.3 Features importance

As shown in the heatmap in Figure 5.2, we analyze the importance of different features in predicting the S&P 500 return using Lasso, Ridge, Elastic Net, PCA, and PLS. The feature importance results for the Lasso, Ridge, and Elastic Net regularization methods appear quite similar. In contrast, PCA and PLS display distinct patterns of feature importance, not only compared to the regularization methods but also in relation to each other.

Figure 5.2 presents feature importance as follows: for PCA, variables are selected based on the amount of variance they explain in the predicted variable. A value of 1 means the feature explains the most variance, while lower values indicate a lesser contribution. In the PLS column, the values show how well each feature explains the predicted variable—in this case, the S&P 500 return. Here, a value of 1 signifies a major role in prediction, while a value of 0 indicates no contribution. PLS assigns importance values between 0 and 1 based on the correlation between each feature and the predicted variable.

For Lasso, Ridge, and Elastic Net, these regularization methods use increasing penalties that gradually shrink some coefficients to zero. In the heatmap, a value of 1 means the feature's coefficient persists the longest as regularization increases, whereas features with values near zero are those whose coefficients shrink to zero more rapidly.

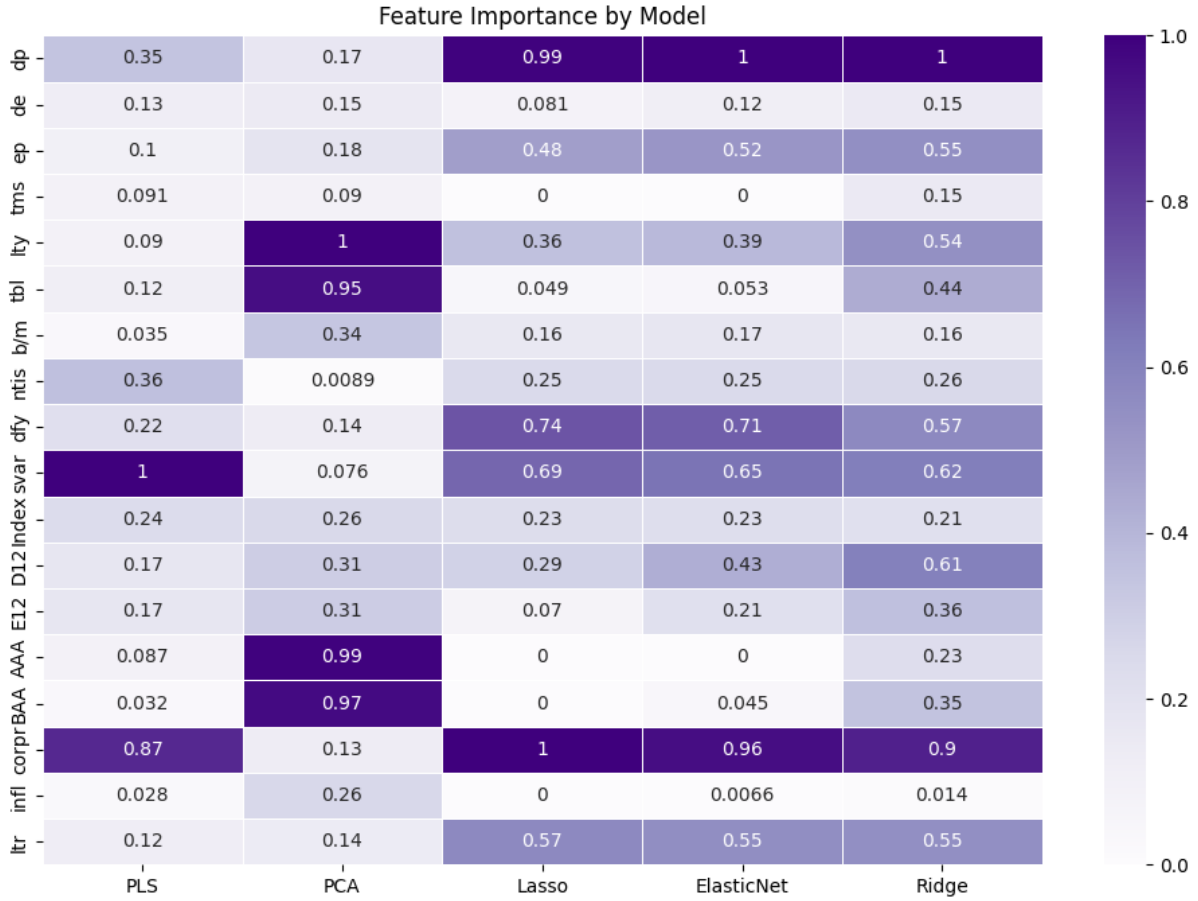

Figure 5.2: Feature Importance by Model for Different Regularization Methods: PLS, PCA, Lasso, Elasti-cNet, and Ridge

5.3.1 Random Forest

To assess the impact of model complexity, we tested different numbers of estimators—corresponding to varying numbers of decision trees—for our out-of-sample forecasts. The results are presented in Table 5.10. As observed, the number of estimators has a noticeable effect on forecast accuracy, with increasing estimators leading to a slight improvement.

Although the random forest models show varying predictive performance, as measured by R^2, all models yield positive out-of-sample R^2 values. However, when applying the Diebold-Mariano test to compare these models with a simple average model, we find that the random forests with 100, 1000, and 5000 estimators all yield p-values significantly greater than 5%. This indicates that we cannot reject the hypothesis that the naïve model, which uses a simple average, performs better in predicting stock market returns. This finding contrasts somewhat with the positive R^2 values observed for these three random forest models.

In their research, Rossi (2018) applied boosted regression trees (BRT) to forecast the S&P 500. Their out-of-sample R^2 reached 0.30%. As shown in Table 5.10, BRT exhibited lower performance. They used the same features as the study of Welch and Goyal (2008) train the BRT model.

Model	**Out-of-Sample R^2**	**DM Test Statistic**	**DM Test P-value**
100 Estimators	0.0389	-0.9257	0.3546
1000 Estimators	0.0464	-1.0816	0.2794
5000 Estimators	0.0496	-1.0553	0.2913

Table 5.10: Out-of-sample R^2 for random forest models with different numbers of estimators.

5.3.2 XGboost

To evaluate the performance of gradient boosting, we used XGBoost, which leverages a large number of weak learners to produce a stronger forecaster. As shown in Table 5.11, our XGBoost model with 500 estimators performed poorly in the out-of-sample period, with an R^2 of -0.2961. This illustrates that this more complex model performed worse than a simple average model in forecasting our index during the out-of-sample period. Because of the poor performance of the XGBoost model in out-of-sample R^2, we did not perform the Diebold-Mariano test to compare it with the naive model.

Model	Out-of-Sample R^2
500 Estimators	-0.2961

Table 5.11: Out-of-sample R^2 for XGBoost with 5000 estimators.

5.3.3 Neural networks

To evaluate the effect of neural network complexity, we employed architectures NN1 through NN5 to examine how increasing the number of layers and neurons influences the predictive accuracy of our model. Our primary objective was to assess whether a more complex neural network structure enhances forecasting performance. As indicated in Table 5.12, more intricate architectures generally yield higher forecasting accuracy during the out-of-sample period. This trend contrasts with the in-sample forecasting results in Table 5.5, where simpler neural architectures with fewer layers achieved better performance. Specifically, single-layer networks performed well in-sample but poorly out-of-sample, while deeper multi-layer networks performed better out-of-sample but worse in-sample. Furthermore, although increasing the number of layers tends to improve the out-of-sample R^2, we observe a slight decline in accuracy from NN4 to NN5, which is difficult to interpret and may stem from the inherent "black box" nature of neural networks.

When comparing our neural network models against the naïve model, which represents a simple average, using the Diebold-Mariano test, we find that NN3, NN4, and NN5—being more advanced models with three, four, and five layers—exhibit p-values below 5%. This indicates their superior performance relative to the naïve model at a 5% significance level. Conversely, NN1 and NN2 yield p-values above 5%, suggesting weaker predictive performance compared to the simple average model. This observation aligns with the general expectation that increasing the number of layers enhances model sophistication and predictive capability. However, this trend is not absolute, as evidenced by our results, where NN5 demonstrates a lower out-of-sample R^2 value compared to NN4 and a higher p-value in the Diebold-Mariano test. This anomaly may again be attributed to the "black box" characteristics of neural networks.

The results of the Diebold-Mariano test and R^2 show that increasing the number of neural architecture layers, and consequently the number of neurons, improves the model's out-of-sample forecasting performance.

Model	Out-of-Sample R^2	DM Test Statistic	DM Test P-value
NN1	-0.1358	-0.7622	0.4460
NN2	0.0277	0.9545	0.3398
NN3	0.0763	-2.2750	0.0229
NN4	0.0925	-2.6946	0.0070
NN5	0.0808	-2.3794	0.0173

Table 5.12: Out-of-sample R^2 for neural network models.

The performance of our NN4 model, which demonstrated the best results over the total out-of-sample period, was further examined by calculating the R^2 of our predictions using rolling 10-year windows, advancing one month at a time. As shown in Figure 5.3, the accuracy of the model fluctuates over time, with its weakest performance observed during the 2008 financial crisis. The R^2 values of the NN4 model exhibit substantial variation across different 10-year periods, ranging from 0 to nearly 30% between 1940 and 2020.

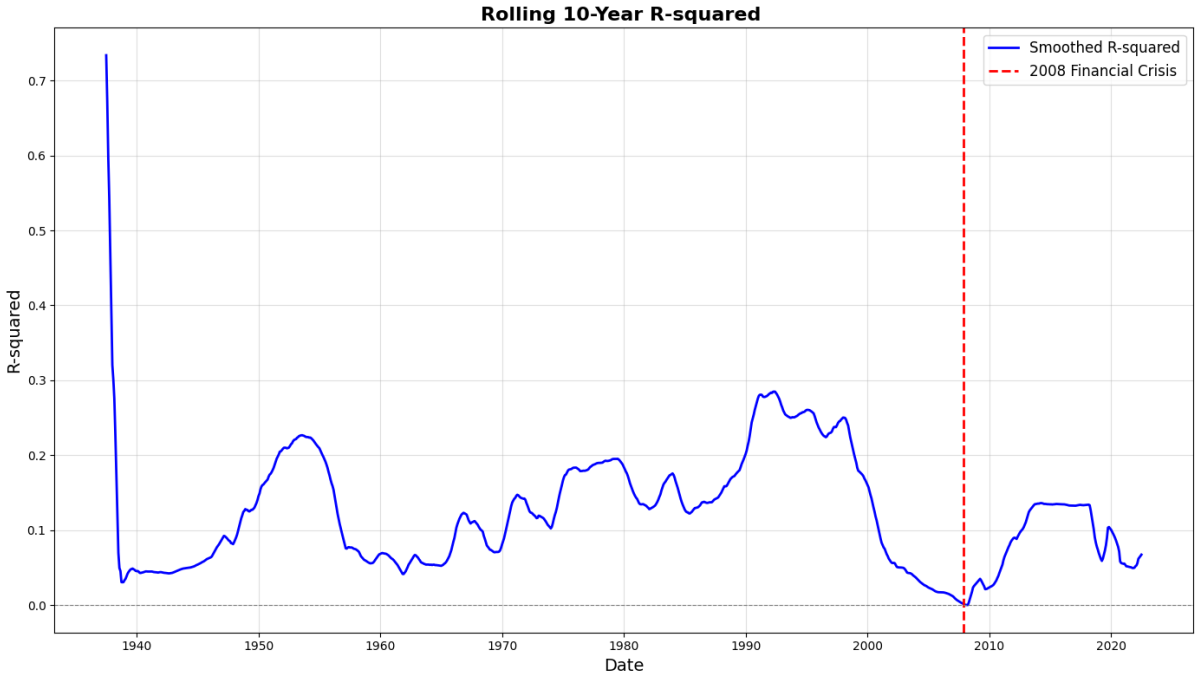

Figure 5.3: Rolling 10-year R^2 between S&P500 returns and out-of-sample predictions of the NN4 model, with a vertical line marking the 2008 Financial Crisis.

5.3.4 Cross validation and Neural Networks

To further explore model performance, we evaluated five different neural architectures (two NN4 models and three NN5 models) across each of our out-of-sample periods. In each rolling window, we selected the model with the highest R^2 value. The results are presented in Table 5.13. As shown, this approach outperforms all the other models we tested during the out-of-sample period, achieving an R^2 of 0.0926. This represents strong performance for out-of-sample forecasting, especially in a volatile market such as the stock market.

Model	**Out-of-Sample R^2**	**DM Test Statistic**	**DM Test P-value**
Through 5 Neural Architectures	0.0926	-2.7490	0.0060

Table 5.13: Out-of-sample R^2 for the best-performing model selected from five neural architectures.

5.3.5 RNN and LSTM

RNN and LSTM are deep learning methods designed to capture long-term dependencies in data. In this analysis, we evaluated their performance by setting aside 25% of the data as a test set. The R^2 values for the test set are presented in Table 5.14. As shown, both models performed poorly during the test period. Due to their weak out-of-sample R^2, we did not conduct the Diebold-Mariano test to compare them with the naive model.

RNN and LSTM are deep and complex neural architectures that typically perform well with large datasets. However, our dataset consists of monthly observations, which are limited in both sample size and number of features. As a result, the models performed poorly in this setting. In fact, for both RNN and LSTM, the out-of-sample R^2 was even lower than -100%, clearly indicating their weak predictive performance with our dataset.

Metric	**Value**
RNN Test R-squared	-2.0070
LSTM Test R-squared	-1.0596

Table 5.14: Test R-squared Values for RNN and LSTM Models

5.4 Robustness

In empirical research, robustness refers to the stability and reliability of results when key elements of a model are changed. Robustness is crucial because models can be sensitive to certain assumptions, parameter values, or data configurations. By performing robustness tests, we can determine whether our findings remain consistent under different modeling conditions or when using subsets of the data. In forecasting, robustness adds credibility to model results and supports their practical use in real-world decision-making.

To assess the reliability of our models, this section presents some robustness checks. We examine how changes to key model parameters—such as regularization strength and training window size—affect the forecasting performance of our models.

5.4.1 Effect of Regularization Strength

We varied the regularization parameter (λ) for both Ridge and Lasso regression models. As shown in Table 5.15, this adjustment had distinct effects on each model. Ridge regression exhibited a gradual improvement in out-of-sample R^2 as λ increased. As the regularization parameter became larger, the R^2 values stabilized. This behavior may occur because all the regression coefficients (β) shrink progressively toward zero. Eventually, beyond a certain threshold of λ, only the intercept remains, which might explain why we still observe a positive R^2 in some cases. In contrast, Lasso's performance declined with higher λ values. The results in the table suggest that there should be an optimal value of λ for our Lasso model. These findings highlight the importance of carefully tuning λ for each dataset, as the optimal value can significantly influence forecasting accuracy.

Model	Lambda (λ)	Out-of-Sample R^2
Ridge	0.0001	-0.1689
Ridge	0.0005	-0.0017
Ridge	0.0010	0.0297
Lasso	0.0001	0.0246
Lasso	0.0005	-0.0529
Lasso	0.0010	-0.0538

Table 5.15: Out-of-sample R^2 values for Ridge and Lasso models using different λ values. This table shows how the predictive performance changes as we adjust the regularization strength for each model.

As shown in Figure 5.4, we plotted the out-of-sample R^2 values against the Ridge regularization parameter λ, for values ranging from 0 to 0.0005. The graph was generated using 50 different points between these λ values, with the model run separately for each value to compute the corresponding out-of-sample R^2. As λ increases, we observe a smooth rise in out-of-sample R^2, although the rate of improvement slows as λ becomes larger.

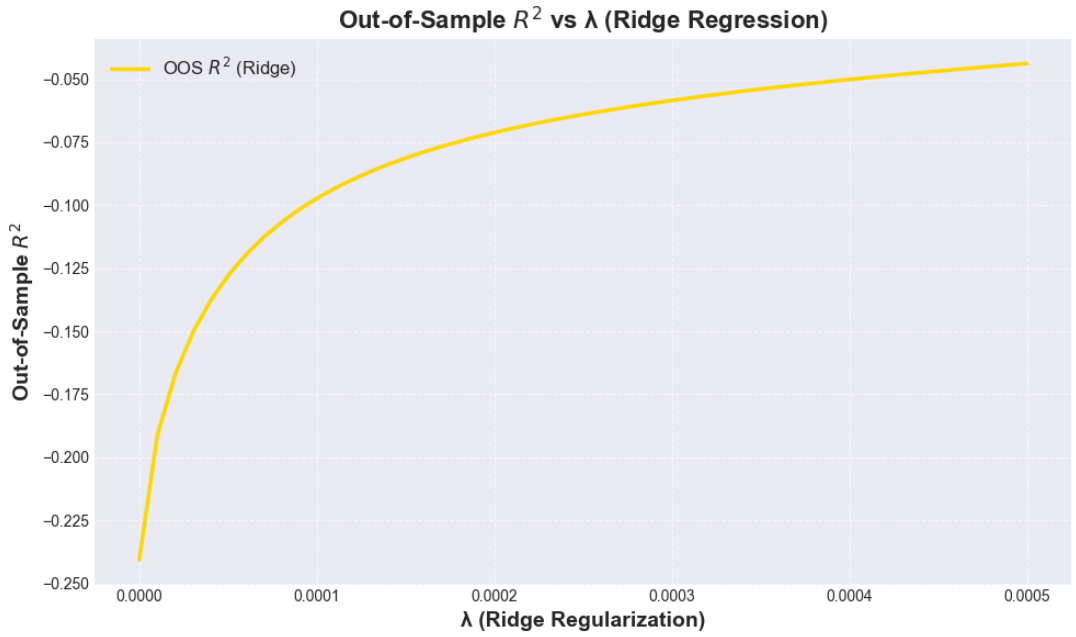

Figure 5.4: Out-of-sample R^2 for Ridge Regression as a function of the regularization parameter λ.

Figure 5.5 presents the results of our Lasso regression analysis. Using the same approach, we generated the graph by calculating out-of-sample R^2 values for 50 different

λ settings. Notably, the sensitivity of the Lasso model to λ is different from Ridge regression. There is a distinct peak in performance, with out-of-sample R^2 reaching its maximum at $\lambda \approx 0.0001$, which matches the value used in our initial regularization experiments. Beyond this point, the model's performance declines as the regularization parameter increases. While $\lambda = 0.0001$ appears to be optimal for our Lasso regression on this dataset, it is important to note that this value may not be optimal for all datasets.

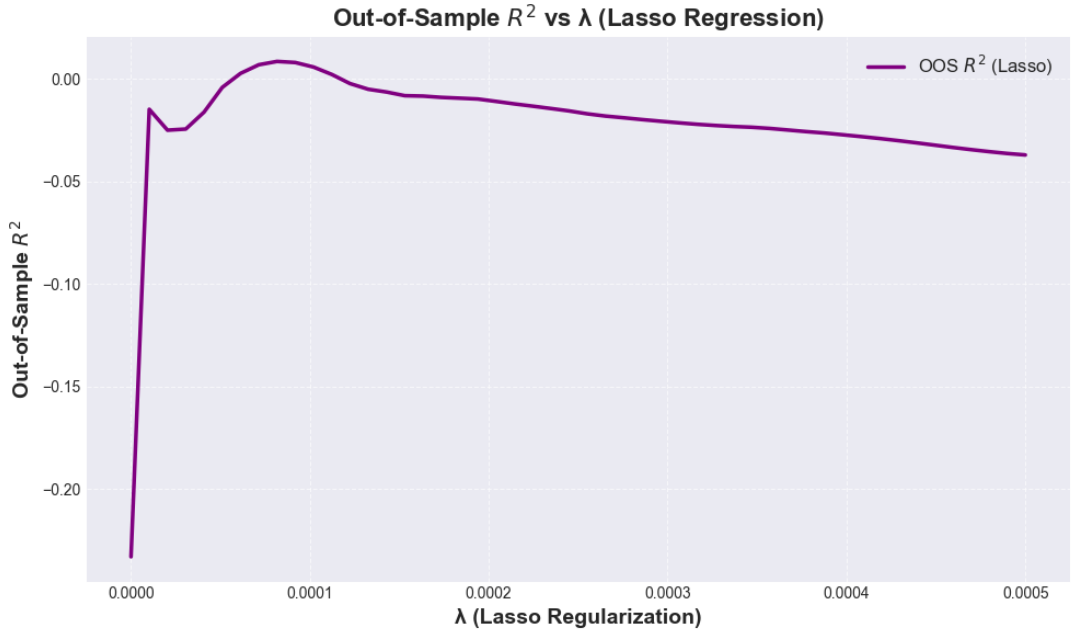

Figure 5.5: Out-of-sample R^2 for Lasso Regression as a function of the regularization parameter λ. The plot shows 50 λ values, highlighting a peak in performance at $\lambda \approx 0.0001$. For larger λ, the Lasso model's performance decreases.

5.4.2 Effect of Rolling Window Size

We also tested the robustness of our Random Forest model by altering the size of the rolling training window. In our baseline setup, we followed Gu et al. (2020) and used a 120-month (10-year) window. To examine the sensitivity of the model to this parameter, we also tested window sizes of 60 months (5 years) and 240 months (20 years).

As shown in Table 5.16, the model's out-of-sample R^2 varied across window sizes. The 60-month window resulted in the poorest performance, likely due to limited training data. Interestingly, the 240-month window also underperformed compared to the 120-month baseline, despite having a larger training sample. This may suggest that older data becomes less relevant over time or that a 10-year window strikes an effective balance between sample size and temporal relevance. These results highlight the importance of choosing an appropriate training window and raise interesting questions about how well such configurations generalize across different datasets or markets.

Rolling Window Size (months)	**Out-of-Sample R^2**
60	-0.0580
120	0.0389
240	0.0291

Table 5.16: Out-of-sample R^2 for Random Forest using different rolling window sizes. The results show how prediction accuracy varies depending on the amount of historical data used for training.

Chapter 6

Coclusion

As we observe, all models exhibit positive R^2 values during the in-sample period for predicting the S&P 500 monthly return. Notably, models such as Random Forest and XGBoost achieve the highest R^2 values in this period. However, it is important to note that these extremely high R^2 values—such as XGBoost with 500 estimators reaching nearly 1 ($R^2 = 0.6886$)—are most likely a result of overfitting. This overfitting in the in-sample predictions helps explain why these models fail to perform as well in the out-of-sample period.

In out-of-sample forecasting, overly complex models such as XGBoost, LSTM, and RNN exhibit negative out-of-sample R^2 values, which stands in stark contrast to their strong in-sample performance. This discrepancy can be attributed to overfitting. A similar trend is observed with Random Forest: although it achieves an impressive in-sample R^2 of over 86%, its out-of-sample R^2 drops to below 5%, highlighting a significant decline. By comparison, neural network models have an average in-sample R^2 of around 20%, and models such as NN3, NN4, and NN5 maintain an average out-of-sample R^2 of approximately 8%, resulting in a much smaller gap between in-sample and out-of-sample performance.

In linear regression and regularization methods (Lasso, Elastic Net, and Ridge), combining these models with Partial Least Squares (PLS) and Principal Component Regression (PCR)—where PCR uses top features from Principal Component Analysis (PCA)—leads to a decrease in in-sample R^2. This reduction occurs because models trained on all features are more prone to overfitting during the in-sample period, while applying PCR and PLS helps mitigate overfitting, resulting in lower in-sample R^2. How-

ever, during the out-of-sample period, combining these techniques with linear regression and regularization does not significantly improve or reduce the model's R^2. Based on this evidence, we conclude that these approaches do not specifically enhance model performance.

Random Forest models perform relatively well in the out-of-sample period in terms of R^2, and their performance tends to improve as the number of estimators increases. For example, training the model with 5000 estimators yields an R^2 of 4.96%. However, according to the Diebold-Mariano test, even at the 80% confidence level, we cannot conclude that our Random Forest models significantly outperform the naive simple average method.

Our neural network models, particularly those with more complex architectures such as NN4 and NN5, achieve out-of-sample R^2 values of 9.25% and 8.08%, respectively. These are the highest out-of-sample R^2 values among all our models. Furthermore, the Diebold-Mariano test indicates that, at the 95% confidence level, both NN4 and NN5 significantly outperform the naive simple average model.

Based on both the R^2 criterion and the Diebold-Mariano test, neural networks demonstrated superior performance compared to our other models during the out-of-sample period. In contrast, while the Random Forest model did achieve a positive R^2 value, the Diebold-Mariano test indicates that even the naive simple average model outperformed it.

Another interesting result from our out-of-sample performance evaluations is that, at first glance, one might expect more complex models to yield more accurate forecasts of stock market returns. However, our findings suggest otherwise. For example, both RNN and LSTM—despite being the most complex models with memory capabilities (including short-term and long-term memory in LSTM)—produced negative R^2 values in the out-of-sample period. Additionally, with neural networks, increasing the number of layers from 4 to 5 actually led to lower performance: NN4 outperformed NN5 in both R^2 and the Diebold-Mariano test. This suggests there may be a threshold where further increasing model complexity no longer improves, and may even harm, forecasting accuracy.

Regarding the robustness of our models, we initially used the window size recommended in Welch and Goyal (2008). We then tested the robustness of our results by varying the window size and found that the 120-month (10-year) window provided the

best performance among those tested. However, since this finding is based on the same market dataset, it does not guarantee that this window size is optimal for other datasets or market conditions.

Regarding the regularization parameter λ, Figure 5.5 shows that $\lambda = 0.0001$ is close to the optimal value, as indicated by the peak in the graph. This result is based on calculations at 50 different values, so the chosen λ may not be the exact optimum, but it is near the best value identified for our dataset. However, it is important to note that this parameter may not be optimal for other datasets or market conditions. Additionally, since we used R^2 as the metric for forecasting accuracy, a different metric could potentially yield a different optimal value for λ.

In my opinion, the underperformance of advanced models such as RNN, LSTM, and XGBoost in the out-of-sample period can be attributed to their need for large datasets to effectively capture complex patterns. Our dataset was relatively limited, both in the number of features (columns) and in the frequency of observations, as we used monthly data. Employing daily data would increase the number of rows and could potentially enhance the performance of these more complex models.

There are also other ways to improve the performance of advanced models such as XGBoost and LSTM. One approach is to combine stock return data from multiple markets to create a larger dataset. However, this strategy presents its own challenges, as different markets may have distinct characteristics, experience recessions at different times, and respond differently to economic shocks. To ensure consistency and improve model reliability, it would be preferable to combine data from markets with similar features.

References

Atsalakis, G. S., & Valavanis, K. P. (2009). Surveying stock market forecasting techniques – part ii: Soft computing methods. *Expert Systems with Applications*, *36*(3), 5932–5941.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth International Group.

Campbell, J. Y. ., & Yogo, M. (2006). Efficient tests of stock return predictability. *Journal of Financial Economics*, *81*(1), 27–60.

Campbell, J. Y., & Thompson, S. B. (2008). Predicting excess stock returns out of sample: Can anything beat the historical average? *The Review of Financial Studies*, *21*(4), 1509–1531.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 785–794.

Chun, D., Kang, J., & Kim, J. (2024). Forecasting returns with machine learning and optimizing global portfolios: Evidence from the korean and u.s. stock markets. *Financial Innovation*, *10*(1), 124. https://doi.org/10.1186/s40854-024-00648-w

Cong, L. W., Tang, K., Wang, J., & Zhang, Y. (2020). Alphaportfolio: Direct construction through deep reinforcement learning and interpretable ai. *SSRN*.

Diebold, F. X., & Mariano, R. S. (1995). Comparing predictive accuracy. *Journal of Business & Economic Statistics*, *13*(3), 253–263.

Ding, H. (2023). Evaluating the effectiveness of elastic net model in predicting stock closing price. *ResearchGate*. https://www.researchgate.net/publication/385543040_Evaluating_the_effectiveness_of_elastic_net_model_in_predicting_stock_closing_price

Gu, S., Kelly, B., & Xiu, D. (2020). Empirical asset pricing via machine learning. *The Review of Financial Studies*, *33*(5), 2223–2273. https://doi.org/10.1093/rfs/hhz033

Hastie, T. (2020). Ridge regularization: An essential concept in data science. *Technometrics*, *62*(4), 426–433. https://doi.org/10.1080/00401706.2020.1791959

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.

Kelly, B., & Xiu, D. (2023). Financial machine learning. *Foundations and Trends® in Finance*, *13*(3–4), 205–363. https://doi.org/10.1561/0500000064

Lewellen, J., Nagel, S., & Shanken, J. (2010). A skeptical appraisal of asset pricing tests. *Journal of Financial Economics*, *96*(2), 175–194.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Neba, C., Nsuh, G., Fuhnwi, G. S., Amouda, P., Neba, A., Adebisi, A., Kibet, P., & Webnda, F. (2023). Comparative analysis of stock price prediction models: Generalized linear model (glm), ridge regression, lasso regression, elasticnet regression, and random forest. *International Journal of Innovative Science and Research Technology*, *8*(3), 14–26. https://doi.org/10.38124/IJISRT23MAR141

Nelson, D. B., & Kim, Y. (1993). Predictable stock returns: The role of small sample bias. *The Journal of Finance*, *48*(2), 641–661.

Nelson, D. M. Q., Pereira, A. C. M., & de Oliveira, R. A. (2017). Stock market's price movement prediction with lstm neural networks. *2017 International Joint Conference on Neural Networks (IJCNN)*, 1419–1426. https://doi.org/10.1109/IJCNN.2017.7966019

Niu, L., Sun, M., Yu, M., & Wang, K. (2022). Point and interval forecasting of ultra-short-term wind power based on a data-driven method and hybrid deep learning model. *Energy*, 124384.

Rossi, A. G. (2018). Predicting stock market returns with machine learning [Available at Smith School of Business, University of Maryland]. *University of Maryland*.

Stambaugh, R. F. (1999). Predictive regressions. *Journal of Financial Economics*, *54*(3), 375–421.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, *58*(1), 267–288.

Wang, J. (2022). The comparison of stock return prediction for random forest, ordinary least square, and xgboost. *BCP Business & Management*, *26*, 262–270. https://bcpublication.org/index.php/BM/article/download/2028/2023

Welch, I., & Goyal, A. (2008). A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies*, *21*(4), 1455–1508. https://doi.org/10.1093/rfs/hhm014

White, H. (1988). Economic prediction using neural networks: The case of ibm daily stock returns. *Neural Networks, 1988., IEEE International Conference on*, 451–458.

Yuan, M., & Lin, Y. (2007). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society, Series B*, *68*(1), 49–67.

Zhang, Y. (2023). Stock price prediction method based on xgboost algorithm. *Proceedings of the International Conference on Big Data and Education Management (ICBBEM 2022)*, *5*, 595–603. https://doi.org/10.2991/978-94-6463-030-5_60

Zhu, Y. (2020). Stock price prediction using the rnn model. *Journal of Physics: Conference Series*, *1650*(3), 032103. https://doi.org/10.1088/1742-6596/1650/3/032103

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, *67*(2), 301–320.

Appendix

Figure 6.1 illustrates the training and testing performance of our RNN model on both the training and testing sets. The model's accuracy was measured using Mean Squared Error (MSE), where each epoch represents one complete pass of the data through the neural architecture from beginning to end. We analyzed the behavior of the RNN model across 100 epochs, meaning the entire dataset was passed through the network and trained 100 times. As shown in the figure, both the training set and testing set MSE decrease during the initial epochs. However, after the 20th epoch, the testing set MSE begins to increase, while the training set MSE continues to slowly decrease toward zero. This pattern suggests the onset of overfitting after 20 epochs, where the model starts to fit the training data too closely and loses its generalization ability on new data.

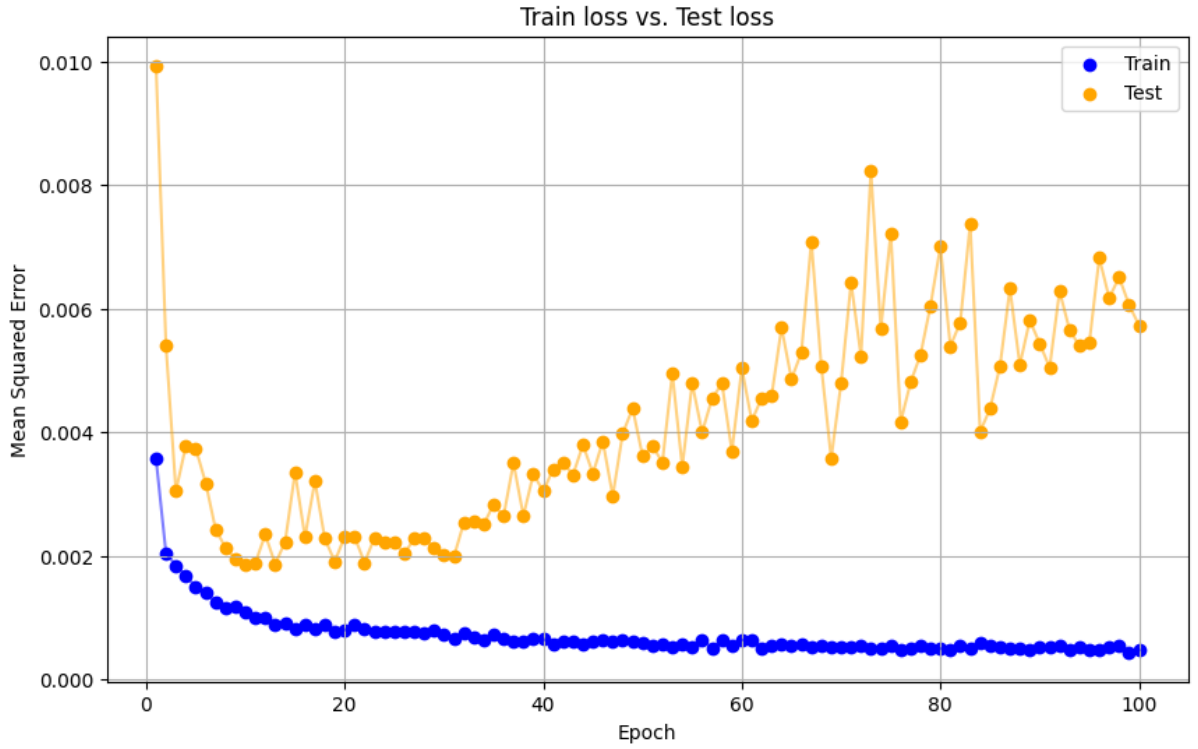

Figure 6.1: RNN: Train Loss vs. Test Loss

Figure 6.2 illustrates the training and testing performance of our RNN model on both the training and testing sets, with accuracy measured by the R Squared Error (R^2). Each epoch represents one complete pass of the data through the neural architecture, from beginning to end. We analyzed the behavior of the RNN model over 100 epochs, meaning the entire dataset was passed through and trained by the network 100 times. As shown in the figure, both the training and testing set R^2 values increase during the initial epochs. However, after the 20th epoch, the testing set R^2 starts to decrease, while the training set R^2 continues to gradually increase toward 1. This trend indicates that the model begins to overfit the training data after around 20 epochs, resulting in reduced performance on unseen data.

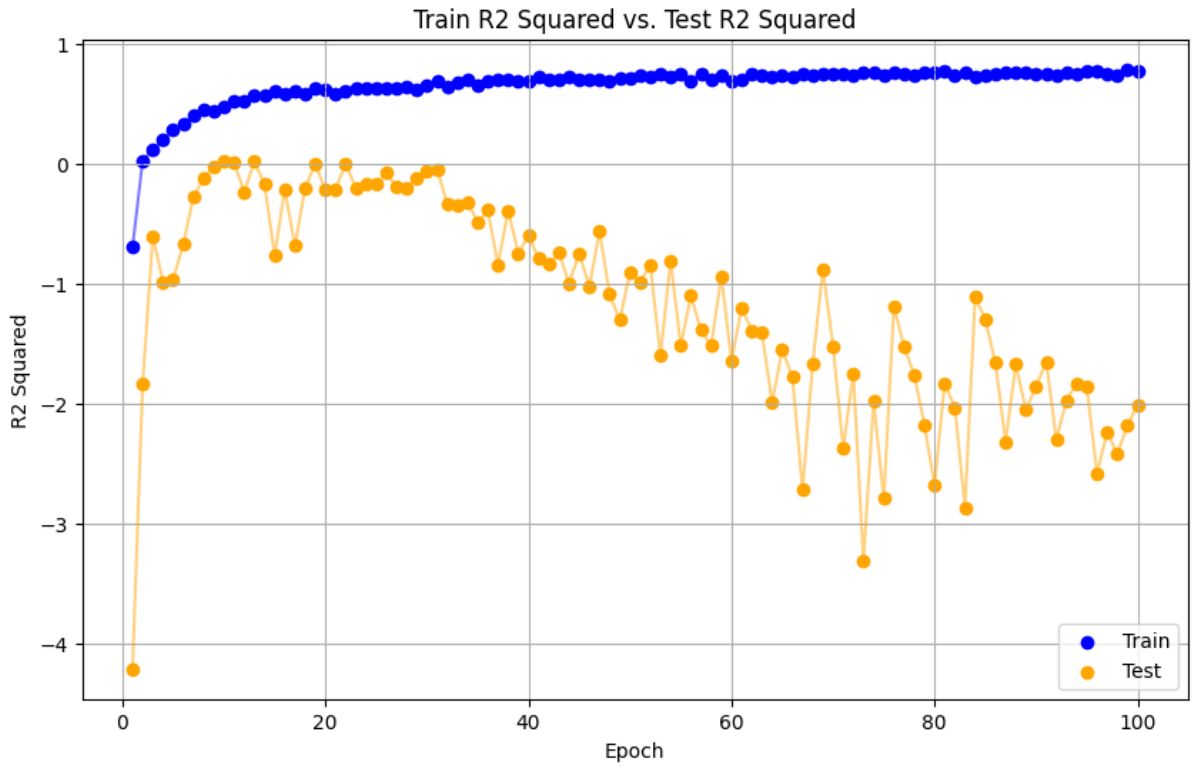

Figure 6.2: RNN: Train R^2 vs. Test R^2

Figure 6.3 illustrates the training and testing performance of our LSTM model on both the training and testing sets. The model's accuracy was measured using Mean Squared Error (MSE), with each epoch representing one complete pass of the data through the neural architecture from start to finish. We analyzed the behavior of the LSTM model over 100 epochs, meaning the entire dataset was passed through and trained by the network 100 times. As shown in the figure, both the training and testing set MSE decrease during the initial epochs. However, unlike previous models, the testing set MSE begins

to increase almost immediately, starting from the first few epochs, while the training set MSE continues to slowly decrease toward zero. This pattern suggests that the LSTM model starts to overfit the training data very early, resulting in a loss of generalization on new data.

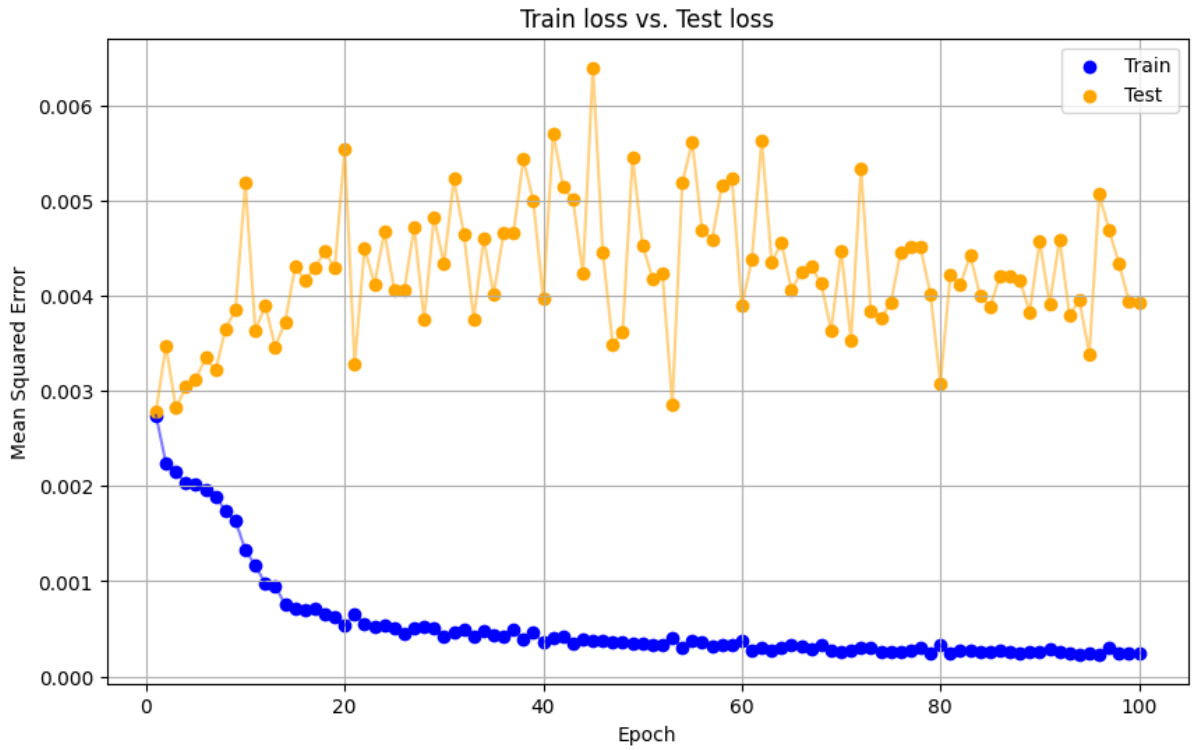

Figure 6.3: LSTM: Train Loss vs. Test Loss

Figure 6.4 illustrates the training and testing performance of our LSTM model on both the training and testing sets, with accuracy measured by the R Squared Error (R^2). Each epoch represents one complete pass of the data through the neural architecture from start to finish. We analyzed the behavior of the LSTM model over 100 epochs, meaning the entire dataset was passed through and trained by the network 100 times. As shown in the figure, both the training and testing set R^2 values increase during the initial epochs. However, the testing set R^2 starts to decrease almost immediately, beginning from the first few epochs, while the training set R^2 continues to gradually increase toward 1. This indicates that the LSTM model begins to overfit the training data very early, leading to reduced generalization performance on the testing set.

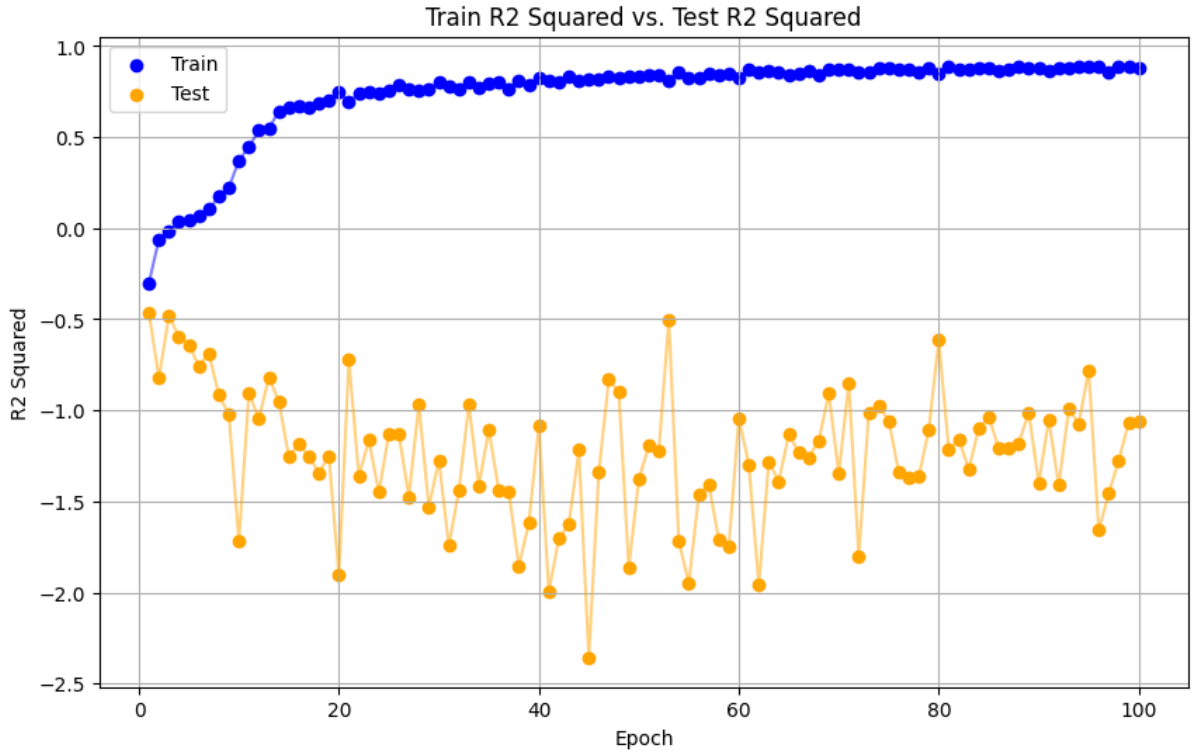

Figure 6.4: LSTM: Train R^2 vs. Test R^2

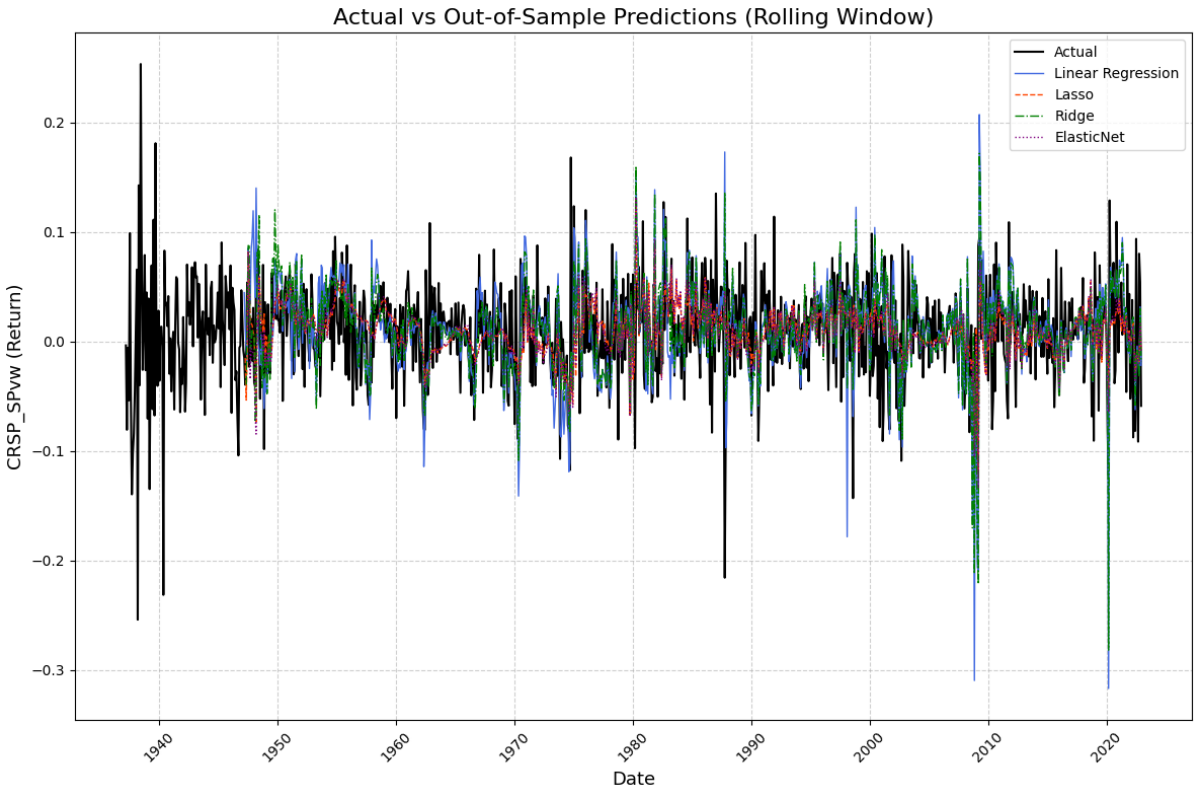

Figure 6.5: This graph shows the out-of-sample predictions of our regression models — Linear Regression, Lasso, Ridge, and ElasticNet — compared to the actual monthly returns of the S&P 500 (CRSP_SPvw). The black line represents the actual returns, while the colored lines show each model's out-of-sample fit over time.

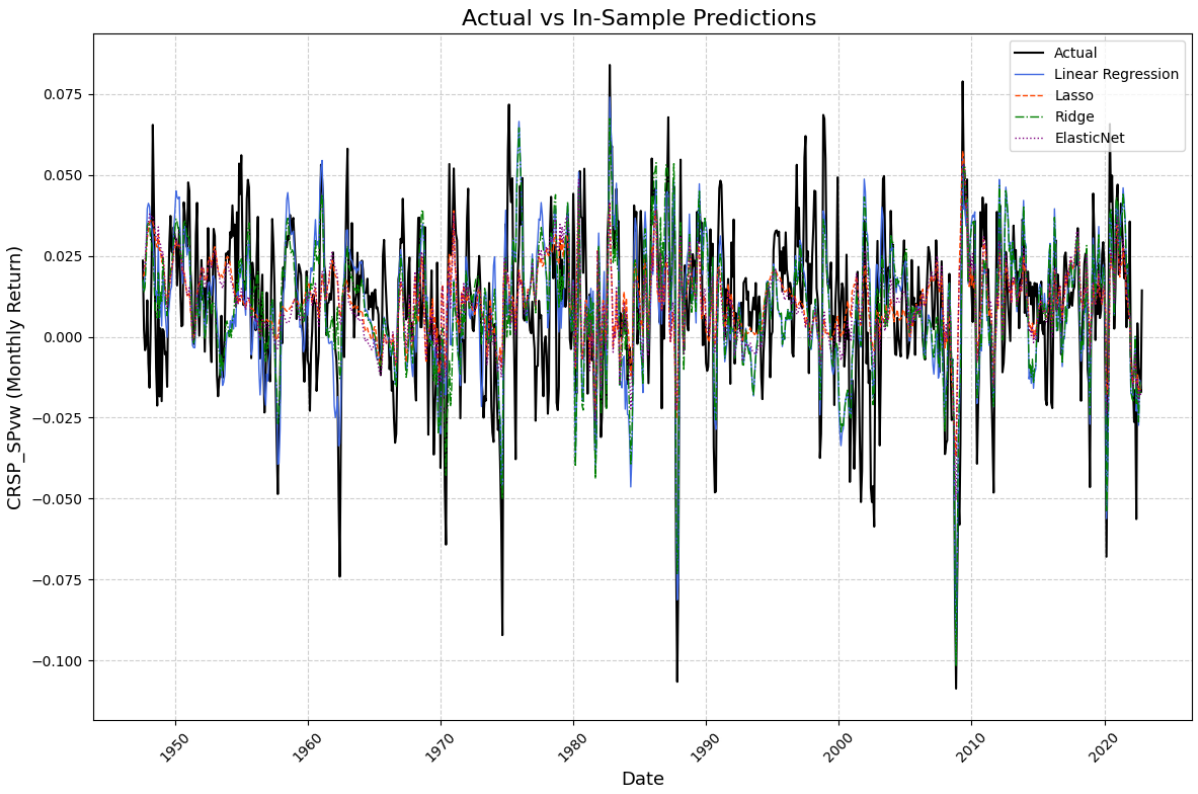

Figure 6.6: This graph shows the in-sample predictions of our regression models — Linear Regression, Lasso, Ridge, and ElasticNet — compared to the actual monthly returns of the SP 500 (CRSPvw). The black line represents the actual returns, while the colored lines show each model's in-sample fit over time.