Dimensionality Reduction and Machine Learning Approaches to Factor Investing and Stock Return Prediction:

A Comparative Study of SPCA versus LASSO Feature Selection on Neural Network

April 19th, 2025

AFM 423, University of Waterloo

Kyle Shi 20933475
Iris Gao 20815552

Table of Contents

Abstract

This project implements two different machine learning methodologies to construct optimal portfolio based on predicted 1-month forward stock returns. The first methodology is LASSO feature selection combined with a Neural Network, and the second methodology is Sparse Principal Component Analysis (SPCA) combined with a Neural Network model. These two machine learning strategies are used to construct top 20% long-only portfolio and compared against the baseline strategy of longing the S&P500 market index. Using a comprehensive financial dataset from 2000 to 2018 that comprises information on 1,207 US listed stocks and their associated 93 firm-specific characteristics, LASSO and SPCA are employed to conduct feature selection, which then is used to train Neural Networks to predict stock returns. The predictive performance is evaluated through out-of-sample testing over the period 2017-2018. Portfolios are constructed monthly by ranking predicted returns and the top 20% of ranked stock are being selected in the long-only strategy portfolio. The experimental result shows that the holding period return for SPCA-NN, LASSO-NN and S&P500 passive portfolio is 105%, 41.4% and 10.10%, respectively; while the Sharpe Ratio for SPCA-NN, LASSO-NN and S&P500 passive portfolio is 0.326, 0.254, and 0.135, respectively; which reveals that the top 20% portfolio favors SPCA-NN when comparing against the LASSO-NN and the baseline S&P500 passive strategy.

Introduction

Predicting stock returns has long been a popular subject in academic finance and investment practice. Traditional factor models such as CAPM and Fama-French model, use simple, linear combinations of a few economic drivers, which limits their ability to fully capture non-linear and complex relationships between a wide array of firm-specific and macroeconomic variables. Recent advancement in machine learning models offer enhance predictive performance and model flexibility by allowing the intake of high-dimensional dataset to uncover hidden relationships amongst many predictors.

Application of ML Approach to Factor Investing

This project explores and analyzes various machine learning approaches in factor investing to construct portfolio with optimal return, specifically Sparse principal Component Analysis (SPCA), LASSO regression, and feedforward Neural Networks.
One of the literature papers reviewed in project one introduced a variant of the traditional PCA known as SPCA, which incorporates the *lasso (L1 penalty) and elastic net (L1+L2 penalties)* to modify principal components with sparse loadings, resulting in zero coefficients for many principal component loadings, thereby reducing the number of explicitly used variables. Another literature paper reviewed is based on the Least Absolute Shrinkage and Selection Operator (LASSO) regression, which is a feature selection method that uses L1 penalty to shrink some coefficients to exactly zero, thereby reducing the complexity of the model. However, LASSO cannot capture non-linear relationships.

Neural Network is also discussed in one of the literature papers, which predicts stock returns due to the capability of the model to capture non-linear relationship and interaction effects. A simple feed-forward neural network architecture is employed to capture complex relationships between inputs and outputs, potentially offering improvements in predictability. The NN model consists of three or more layers, where the input layer comprised a set of firm characteristics, then one or more hidden layers capture the interactive effects between different variables and perform non-linear transformations through activation functions on input variables, and the output layer aggregates all information from the hidden layers to generate an output. The literature paper suggests MSE as the preferred objective function and ReLU as the activation function due to its computational simplicity.

In this project, the challenge of high feature dimensionality is addressed by employing two different methods: LASSO regression and SPCA. LASSO performs feature selection across predictors by shrinking some coefficients to exactly zero for those with low predictive power, while SPCA extracts sparse, interpretable latent factors that captures the most variance. These two techniques focus on extracting most informative signals while lower the risk of overfitting. The next step is to implement a feedforward Neural Network for each of the dimensionality reduction technique. After predicting 1-month forward returns for individual stocks, the model outputs are used to rank stocks and construct long-only portfolio based on top 20% predicted performers each month. A baseline passive investing portfolio is also constructed to hold S&P500 over the same investing period. The performance of the machine-learning-based portfolios is evaluated on the test set and then compared against the performance of passive portfolio.

Several caveats must be taken into consideration when applying these models to financial data. Financial returns data is extremely noisy and is characterized by a low signal-to-noise ratio, meaning that the predictive relationships are often weak due to substantial randomness. Financial returns are also subjective to regime shifts and market shocks, which makes model prediction extremely challenging. Since there are many characteristics that can be used as predictors, without appropriate dimensionality reduction, models are highly susceptible to overfitting and fail to generalize to unseen data.

Research Questions and Objective

The goal of this research project is to investigate whether dimensionality reduction combined with a machine learnings model can enhance the ability to construct an optimal portfolio that can yield a superior return. Following the in-depth analysis from project one, where several literature papers that deploys different machine learning models to enhance the factor selection and portfolio construction process were compared in terms of each of their strengths and weaknesses; this research project replicates the models from those literature papers by generally following the implementation plan developed in project one report, with some tweaks and improvements in the implementation process that will be discussed in details in the methodology section. The different combinations of dimensionality reduction and predictive modelling approaches is compared to answer the following research questions that guides the report:

- Can machine learning models built on SPCA or LASSO selected features generate superior investment performance compared to passive investment on S&P500 index over the same period?

- Does dimensionality reduction using SPCA improve stock return predictability relative to direct feature selection via LASSO?
- Does Neural Network provide added value to portfolio construction?

By addressing the first question, it provides direct evidence of whether feature selection and nonlinear modelling technique offers additional value beyond simple market exposure. With the industry being increasingly dominated by the passive investing strategies, leveraging machine learning models that demonstrate an edge over passive benchmarks is valuable and useful. By examining the relative advantages of SPCA versus LASSO in enhancing portfolio performance addresses the question of which method is best at reducing dimensionality in financial machine learning application, which may provide a pathway toward more robust predictive models. The third question explores whether Neural Networks' ability to capture nonlinearities can provide additional value in constructing a more profitable portfolio. All three questions contribute to the broader understanding of how machine learning techniques can be adapted to improve portfolio construction, as well as addressing this research project's objective of comparing which dimensionality reduction model is more effective and whether the combination with Neural Network can enhance factoring investing and improve stock return prediction.

Data Processing

This project involves two data sources: the first data source is the data_ml dataset provided in this course, and the second data source is the S&P500 index (^GSPC) from Yahoo Finance. The data_ml dataset comprises information on 1,207 stocks listed in the US with time periods starting from 1998/11/11 to 2019/03/31 for a total of 244 monthly observations. Besides the stock id and date columns which are used as identifiers, this dataset also contains 93 firm characteristics associated with each of the stocks and 4 labels which are the 1-month, 3-month, 6-month and 12-month forward returns. The returns are total returns that incorporate potential dividend payments over the considered periods.

The data wrangling process begins with selecting the data from period 1999-12-31 to 2019-01-01 and dropping potential target leaking predictors. Extracting data from the period 2000 to 2018, inclusive, to ensure reliability and completeness of the dataset. This timeframe was chosen because several stocks exhibit missing information beyond 2019. The response variable of interest in the project is "R1M_Usd", which is the 1-month forward return. Besides the target variable, the target leaking variables are identified to be "R3M_Usd", "R6M_Usd", and "R12M_Usd". These predictors introduce forward-looking information, in this case, these three variables all represent forward returns, which must be removed to ensure that the model relies only on information available at the time of prediction to avoid target leakage.

Observations (rows) with missing values across predictors are dropped. Duplicated rows are removed as well. Note that columns with zero variance needs to be removed as they carry no predictive information and may cause instability during standardization. Fortunately, no constant columns have been identified, therefore all columns are preserved. The next step is to check whether the predictors need to be standardized. The calculation shows that the mean range of the dataset is between 0.0127 and 0.506, while the standard deviation range is between 0.176 and 0.289. Prior to modeling, the predictors exhibit a wide range of means and standard deviations vary across features. Ideally, the means should be centered around 0 and standard deviations should be around 1, since models like LASSO, SPCA and NN are sensitive to the scale of input variables. Without standardization, predictors with larger magnitudes can disproportionally influence model outcomes, biasing feature selection in LASSO, dominating the identification of principal components and destabilizing gradient optimization in Neural Networks, therefore standardization is applied to all predictor variables except the target variable.

To avoid lookahead bias, the dataset is split into training, validation and test sets chronologically. The training set contains data from 2000 to 2013, which is used to fit the models. The validation set contains data from 2014 to 2016, which is used to tune hyperparameters. The test set from 2017 to 2018 is used for final evaluation of models and portfolio performance on unseen data. This splitting ensures that only past information is used in the training set to predict future returns, while the test set only contains truly unseen data.

The S&P500 data is used separately as a baseline passive investing strategy. From the tidyquant package, the monthly adjusted S&P500 prices from 2017 to 2018 are pulled from Yahoo Finance. The adjusted price is used because it accounts for stock splits, dividends and corporate actions, which reflects the true economic return. The time frame chosen is the same as the time frame for the test set, since ultimately the portfolios being compared holds assets over the period 2017-2018.

Variables and Measures

The 93 predictor variables in this study consist exclusively of firm-specific characteristics. These features capture different economic and financial aspects of individual stocks which include the following:

- Valuation metrics such as price-to-book ratio, price-to-earnings ratio, EBITDA and enterprise value to EBITDA that describes how the market values the company's assets and earnings
- Size and liquidity indicators such as market capitalization, trading volume, volatility measures that reflect the company size and stock liquidity.
- Profitability metrics such as return on assets, return on equity, operating margins, and free cash flow margins, which assess a company's ability to generate profit and cash flows.
- Leverage and capital structure metrics such as debt-to-equity ratio, total debt to capital and net debt, which measures financial risk.
- Momentum indicators such as past 11-month momentum, which captures price trends.

All these predictors are standardized and fed into the LASSO and SPCA model. Many of these variables are highly correlated, and how these two techniques handle highly correlated variables will be discussed in later sections. The response variable is the 1-month forward stock return (R1M_Usd), which already accounts for dividends, stock splits and other corporate actions.

The ultimate objective of this project is not just to predict stock returns, but to construct optimal portfolios based these predictions, several portfolio performance metrics are used to evaluate the effectiveness of each model.

- The average monthly return is the simple mean of all predicted monthly portfolio returns over the test period
- Holding period return is a compounded return over the test period and it is calculated as $HPR = \Pi_{\mathrm{i=1}}^{\mathrm{n}}(1 + r_i) - 1$ where i is the predicted return in month i. This metric captures the total growth in stock return since the beginning of the test period
- Standard deviation of monthly returns measures the volatility of monthly returns. The higher the volatility, the greater the risk.
- Monthly Sharpe Ratio assesses the risk-adjusted returns, which is calculated as $SharpeRatio = \left(E\left[R_p\right] - R_f\right)/\sigma_p$. Note that the calculation of Sharpe Ratio in this project simplifies to $SharpeRatio = E\left[R_p\right]/\sigma_p$ because of two reasons. The first reason is that the US risk-free rates were very low during 2017-2018, roughly around 2%-2.5% annually, which translates to roughly 0.2% per month. This small amount is insignificant and is negligible. The second reason is that dropping this risk-free rate is a common simplification technique in research projects like this when the focus is on relative performance, so it is acceptable to just assume the risk-free rate is 0.

Experimental Methodology

LASSO Feature Selection Process

After chronologically splitting the full dataset into training, validation and test sets, the target returns variable forms the y_train, y_val, and y_test dataset, while the rest of predictor variables form the x_train, x_val, and x_test dataset. Prior to predicting using machine learning models, LASSO regression was applied using 'glmnet' to perform feature selection with 'R1M_Usd' as the response variable. The LASSO model imposes an L1 penalty on the coefficient to shrink those with low predictive power to exactly zero, requires the input of the hyperparameter value lambda, which controls the degree of penalization. In order to determine the optimal lambda value, cross-validation with a default 10-folds was performed on the training set by using 'cv.glmnet' function. Upon obtaining the optimal lambda value, use this value to perform LASSO regression, which would result in a model that retained a number of non-zero coefficients out of the initial 93 characteristics.

In this study, LASSO was applied solely for feature selection to reduce dimensionality, and stock return predictions were subsequently conducted using a feedforward Neural Network model trained on these LASSO selected features. Neural Network model training procedure using LASSO selected feature inputs is discussed in the Neural Networks section below.

SPCA Implementation

The second dimensionality reduction or feature generation technique deployed is SPCA, which constructs new and compressed feature representations for the NN model. The SPCA input involve all 93 standardized firm-specific features, then it seeks to extract a set of orthogonal principal components that depends only on a subset of input variables. Unlike traditional PCA, sparsity is introduced into the loading vectors to enhance interpretability and reduces overfitting risks, therefore SPCA was chosen over traditional PCA to achieve a sparser set of principal components with heavy loading only on a subset of significant variables.

To construct the SPCA, the function spca() is used from the PMA package, with hyperparameters being pre-specified. The number of components extracted K is set to 5, which balances the need for sufficient information while maintaining a manageable number of features. The sparsity level hyperparameter (para) encourages greater sparsity for lower values. Several values of para including para value of 0.2, 0.4, 0.6, and 0.9 are each used as the input to perform SPCA dimensionality reduction to assess the impact on the resulting SPCA loadings and component structure. Based on the resulting loadings, the most optimal para value is chosen that can balance between stability and sparsity. This chosen para value is then used in the SPCA model to generate a new set of features.

In this study, SPCA was applied solely for feature generation, and stock return predictions were subsequently conducted using a feedforward Neural Network model trained on these SPCA generated features. Neural Network model training procedure using SPCA generated features is discussed in the Neural Networks section below.

Neural Network Architecture

LASSO + Neural Network Hyperparameter Tuning

After selecting a subset of features from LASSO regression, a feedforward Neural Network (NN) model was constructed to capture non-linear interactions amongst variables. To achieve the best model performance, a hyperparameter tuning process is required to determine the optimal parameter values to be inputted into the model:

- 1st hidden layer units represent the number of neurons in the first hidden layer, which controls the model's ability to learn complex feature interactions
- 2nd hidden layer units represent the number of neurons in the second hidden layer, which compresses and refines the information learned from the first layer
- Learning rate controls the step size in optimization process. A smaller value leads to slower model weights updates, meaning that the model learns slower and results in more stable convergence.
- batch size is the number of samples used in each smaller batches during training.

The first step of the tuning process was to set a reasonable range for each of the above hyperparameters. For instance, units for 1st layer may take on any value in (16, 32, 64), units for 2nd layer may take on any value in (8, 16, 32), learning rate can be 0.001 or 0.0005, and batch sizes can be either 64 or 128. For all possible combinations within the reasonable value range, the NN model is performed on training data with features selected by LASSO. Each candidate model was trained for 100 epochs, and performance was evaluated on the validation set based on the mean squared error (MSE). After obtaining the optimal hyperparameter combination, the NN model is trained based on these hyperparameters.

As for training control parameter, each epoch represents one full pass through the entire training dataset, the NN model needs to train for enough epochs to gradually learn and minimize loss function. The model is trained for 100 epochs, which is a common value because it is large enough to provide enough room for learning and converge toward a minimum, and at the same time, 100 epochs is short enough to avoid overfitting.

The training and validation loss curves generated by the Neural Network model is plotted to assess the model learning capability, and how well the model is trained on the validation set comparing to the training set. Then the model prediction performance is assessed on both training and test set, where training and test RMSE values are compared to assess the predictive power of the model and whether overfitting issue occurs. A scatter plot between stock returns in test data and predicted stock returns is produced for a sanity check to evaluate whether the model prediction makes sense in a financial modeling setting.

SPCA + Neural Network Hyperparameter Tuning

Following the feature generation step using SPCA, the new features are fed into a feedforward Neural Network for stock return prediction. Although hyperparameters were tuned for the LASSO model, it is necessary to tune again using SPCA features. This is because LASSO uses original predictors, but SPCA generates transformed features, the change in data inputs would affect the training dynamics as the nature and distribution of the inputs are different, therefore, tunning prior to training NN is required. Following a similar tunning process as before such that the units for 1st layer may take on any value in (16, 32, 64), units for 2nd layer may take on any value in (8, 16, 32), learning rate can be 0.001 or 0.0005, and batch sizes can be either 64 or 128. For all possible combinations within the reasonable value range, the NN model is performed on training data with features generated by SPCA. Each candidate model was trained for 100 epochs, and performance was evaluated on the validation set based on the mean squared error (MSE). After obtaining the optimal hyperparameter combination, the NN model is trained based on these hyperparameters.

The training and validation loss curves are plotted to assess model generalization and risk of overfitting. Then the model prediction performance is assessed on both training and test set, where training and test RMSE values are compared to assess the predictive power of the model and whether overfitting issue occurs. A scatter plot between stock returns in test data and predicted stock returns is produced for a sanity check to evaluate whether the model prediction makes sense in a financial modeling setting.

Portfolio Construction and Evaluation

Three long-only portfolios are constructed in this project to compare which method yields the best portfolio performance. For the baseline passive portfolio, the S&P500 index is held over the test period (2017-2018) using financial market data. For the other two portfolios, after training the NN models, for the predicted returns generated by both LASSO+NN and SPCA+NN models, stocks were ranked based on their predicted returns using a percentile ranking. To construct long-only portfolios, select stocks in the top 20% percentile each month and take an equal weight within selected stocks to calculate the average realized returns. The monthly long-only portfolio returns are aggregated over the test period (2017-2018).

Performance metrics such as holding period return, monthly return standard deviation and Sharpe Ratio are then calculated across three portfolios to determine whether machine-learning models have an advantage over the passive strategy, as well as comparing which machine-learning model has a superior performance.

Results and Discussion

LASSO Feature Selection + Neural Networks Result

LASSO Feature Selection Result

The cross-validation result is shown below:

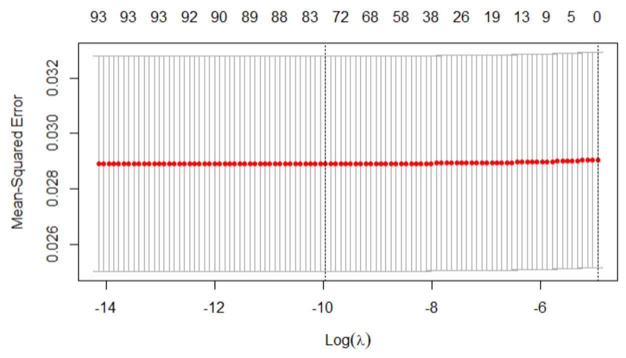


Figure 1 lambda cross-validation result

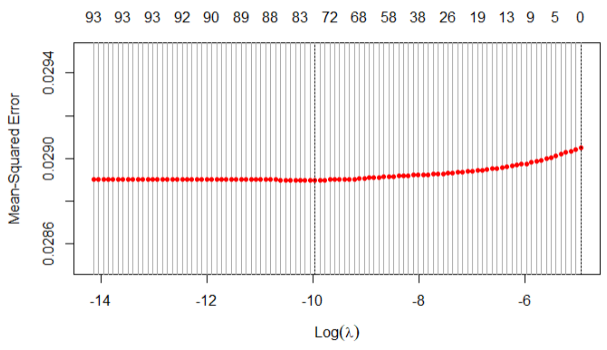


Figure 2 lambda cross-validation result (zoomed-in)

This plot depicts the cross-validation MSE on y-axis and log(lambda) on x-axis. The red dots plot the MSE of each corresponding log(lambda), while the vertical dashed line indicates the log(lambda) value that correspond to the minimum MSE. Note that the above plots are the same except that the right plot is zoomed in, with y-axis between 0.0285 and 0.0295, which can better display the change in MSE. The left plot shows that the MSE curve appears to be relative flat across a wide range of lambda values, suggesting that the model is not highly sensitive to the choice of lambda. From the right plot, there is an obvious trend that the MSE curve starts to increase for log(lambda) greater than -10, and the MSE stays flat for log(lambda) less than -10. Therefore, the optimal lambda value selected would be 0.00004720529, which yields the lowest cross-validation error.

Using the optimal lambda, the LASSO regression was performed to conduct feature selection, which results in a model that retained 81 non-zero coefficients out of the initial 93 characteristics. The LASSO retained a majority of features with similar coefficients, suggesting that these firm-specific variables provide incremental predictive information about the target return but none of which shows significant predictive power. After sorting the coefficient value, the market capitalization and free cash flow on book value have a relatively higher coefficient, which in standardized LASSO, may indicate greater predictive contribution to the model, but it is by no means a direct measure of predictive power. Typically, LASSO would select one representative variable from each correlated group of predictors, however, the selected list retains many variables that are correlated. This might be because these variables contribute to weak but complementary signals. The relatively large set of selected predictors suggests that the financial data is every noisy, a broad predictor set may be needed to extract useful predictive signals.

LASSO + NN Model Prediction Result

The resulting optimal hyperparameter combination is obtained to be 16 units for 1st layer, 8 units for 2nd layer, a learning rate of 0.001, and batch size of 128, which corresponds to the lowest validation MSE of 0.0106.

Using the tuned hyperparameters, the NN model was re-trained on the training set with a model architecture consists of the following:

- an input layer corresponding to the 81 LASSO selected features
- a 1st hidden layer with 16 neurons and ReLU activation as recommended in the literature paper reviewed in project one
- a 2nd hidden layer with 8 neurons and ReLU activation as recommended in the literature paper reviewed in project one
- an output layer with a single neural representing the predicted forward stock return.

The training and validation loss curves for the Neural Network plot is shown below:

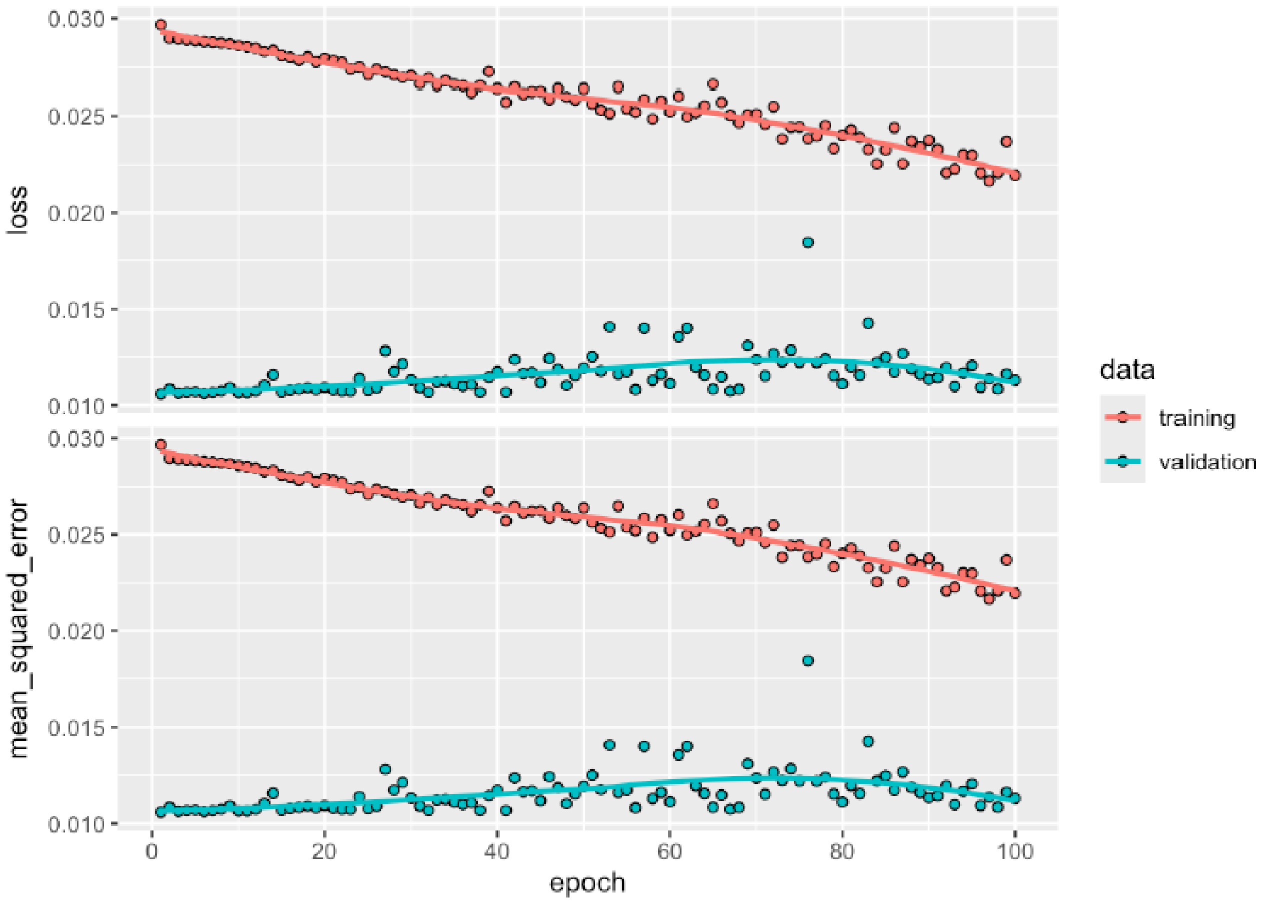


Figure 3 - LASSO+NN Training and Validation Loss Curve

The y-axis represents the mean squared error, and the x-axis represents the number of epochs. The red dots represent training loss curve over epochs while the green dots represent validation loss curve over epochs. Starting from 0 epochs, both training and validation loss steadily decrease across epochs, indicating that the model was picking up meaningful patterns. Note that the initial decrease in training MSE was more stable for less than 40 epochs, then as epoch increases, the training MSE decreases relative stable but with mild fluctuations. The validation MSE remains relatively flat until epoch reaches 40, mild fluctuations (higher MSE) in validation loss toward later epochs, reflecting inherent noise in stock return prediction. On a positive note, the validation MSE fluctuations are contained and do not show a specific upward trend, suggesting that the model is not overfitted. Although the training loss curve is above the validation loss curve across all epochs, the gap between training and validation loss remains relatively small. The small difference of roughly 0.01-0.02 suggests that the model generalizes well to unseen data within the training period. Although it is uncommon for validation MSE to be consistently below the training MSE, it is still explainable as the training set over 2000-2013 period is subject to dramatic economic regime shifts (great financial crisis) whereas the validation set over 2014-2016 may be statistically slightly easier to predict due to stable bull market. Overall, using LASSO selected features and tunned hyperparameters to train the NN achieves a stable learning behavior and a reasonable generalization to the validation set without severe overfitting.

Following NN model training, the model's predictive performance needs to be evaluated separately on the training set and the test set. The validation set is not used for model evaluation because it is used before training to tune the parameters and during training to guide model selection and prevent overfitting. Using the tuned NN model to predict the 1-month forward stock returns on the training and test set. The resulting training RMSE is 0.1466643 while the test RMSE is 0.2803473. The training RMSE measures how well the model predict the training data, and the test RMSE assess the model's ability to generalize to unseen data. The training RMSE of 0.1466643 indicates that the model can capture patterns with reasonable accuracy and without underfitting as the error is not huge. As expected, the test RMSE is higher than the training RMSE, which reflects the challenges of predicting stock returns. The difference in RMSE is noticeable but not extreme, suggesting potential mild overfitting as the test error is almost double the training error; however, since the financial returns are inherently noisy and difficult to predict, the test RMSE is still within reasonable range.

To complement the RMSE evaluation, the actual test returns are plotted against the predicted test returns.

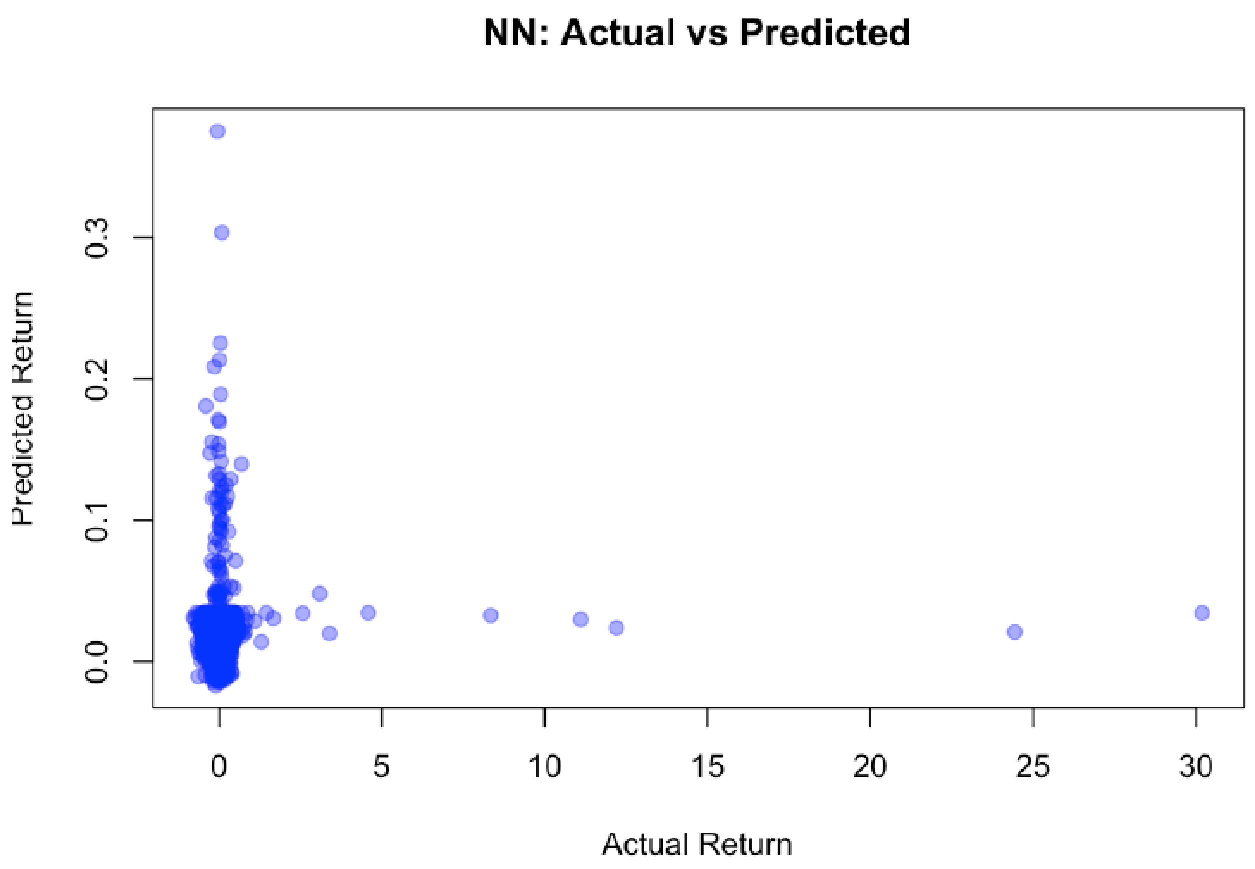


Figure 4 - LASSO+NN Actual vs Predicted Stock Returns

This scattered plot visually displays how well the NN model predicts the return and captures the variability in the target variable. Most points are concentrated and clustered around zero regardless of actual return values, with some negative predictions and some positive predictions between 0.0% and 0.3%. In contrast, the actual stock returns are heavily right-skewed, with a couple of extreme outliers being around 10%, 25% and 30%. The NN model underestimates the magnitude of returns. This conservative prediction is common in financial modeling and is expected because NN model tends to avoid overreacting to extreme observations. Since the focus is on portfolio construction, and less focused on being able to predict precise returns, the absolute accuracy of the predicted returns in this case is less critical than the relative ranking amongst stocks. In the later portfolio construction step, correctly predicting the ranking of the most promising stocks is crucial to the long-only portfolio strategy. Even though the graph shows more compressed and conservative predictions with a limitation in predicting extreme returns, the model's ability to perform stable ranking differentiation can still drive successful stock selection.

SPCA Implementation + Neural Networks Result

SPCA Feature Generation Result

Several values of para (0.2, 0.4, 0.6, 0.9) were tested to evaluate the impact on loadings and the resulting component structure (note that the testing result on different para values are not included in the r-code). The result shows that across all para values, the SPCA loadings remain identical up to the fourth decimal place, suggesting that changes in penalty strength have little impact on extracted components due to highly correlated variables. Based on this finding, a para setting of 0.4 was adopted because this value provides a balance between sparsity and stability. Since para value does not alter the result significantly, then hyperparameter tuning for SPCA is not necessary.

The final loadings for five principal components are listed in the R output, with approximately 62% of loadings being less than 0.05, suggesting that SPCA can successfully shrink the influence of many weaker predictors toward zero. Although none of the SPCA coefficients are forced to exact zeros, this method still effectively prioritized meaningful predictors and suppressed noise from over half of the less informative variables.

Following the extraction of principal components, the original standardized features are multiplied by the SPCA loading matrix to transform the high-dimensional data with 93 predictors into a lower-dimensioned representation based on 5 principal components. The resulting transformed datasets consists of 5 features, each feature being the linear combination of the original predictors with many weaker predictors having minimal influence due to near zero coefficients. By working with the new SPCA components, multicollinearity is handled, risk of overfitting is reduced, and interpretability is improved. Now the transformed features serve as the new inputs for training the NN model, which is discussed in detail in the NN section below.

SPCA + NN Model Prediction Result

The resulting optimal hyperparameter combination is obtained to be 32 units for 1st layer, 8 units for 2nd layer, a learning rate of 0.001, and batch size of 64, which corresponds to the lowest validation MSE of 0.0106.

Using the tuned hyperparameters, the NN model was re-trained on the training set with a model architecture consists of the following:

- an input layer corresponding to 5 SPCA generated features
- a 1st hidden layer with 32 neurons and ReLU activation as recommended in the literature paper reviewed in project one
- a 2nd hidden layer with 8 neurons and ReLU activation as recommended in the literature paper reviewed in project one
- an output layer with a single neural representing the predicted forward stock return.

The training and validation loss curves for the Neural Network plot is shown below:

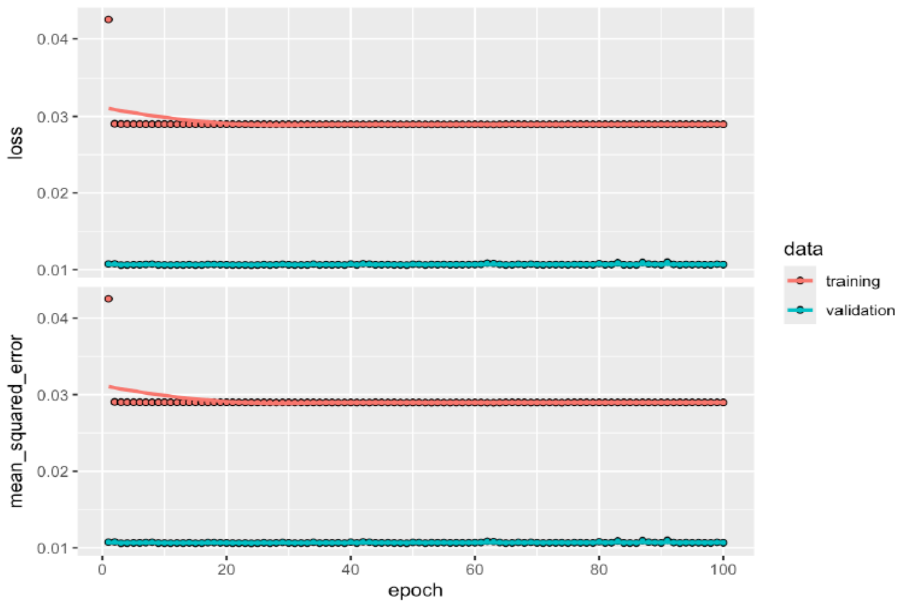


Figure 5 - SPCA+NN Training and Validation Loss Curve

The red dots represent training loss curve while the green dots represent the validation loss curve. Both training and validation loss decreases initially for small epoch, then remained relatively flat throughout the epochs. Furthermore, training loss is roughly 0.03 whereas the validation loss is roughly 0.01, the loss difference of 0.02 remains relatively constant across epochs. This shows that the model generalizes well to unseen data and does not suffer from overfitting. The plot does not show any spikes or oscillations amongst dots, suggesting that the learning rate was appropriately tuned for the SPCA features.

Then the SPCA+NN model is used to predict stock returns based on training and test sets. The RMSE is used as performance metrics to evaluate the model performance. The resulting training RMSE and test RMSE is 0.1702056 and 0.2802199, respectively. The training RMSE is relatively low, indicating a good training fit. However, the test RMSE is somewhat higher, which is within expectation due to the inherent noise and high unpredictability of financial returns. Nonetheless, the gap between test and training RMSE is not huge and within reasonable range, suggesting that the model generalizes reasonably well.

To further investigate model behavior, a scatter plot of actual return versus predicted return on test set is generated. The resulting plot shows that majority of the predicted returns are clustered near zero, while the actual returns are right-skewed, with some outliers around 10%, 25% and 30%. The predicted returns range from 0.00% to 0.10% which underestimates the actual returns. This conservative prediction is normal since the model is trained to minimize MSE, thus it tends to predict values close to the mean zero rather than matching the extreme outliers. Same as the explanation provided for LASSO, the relative ranking of predicted returns is more critical than the accuracy of return predictions. Overall, the plot makes sense given the noisy nature of the financial data.

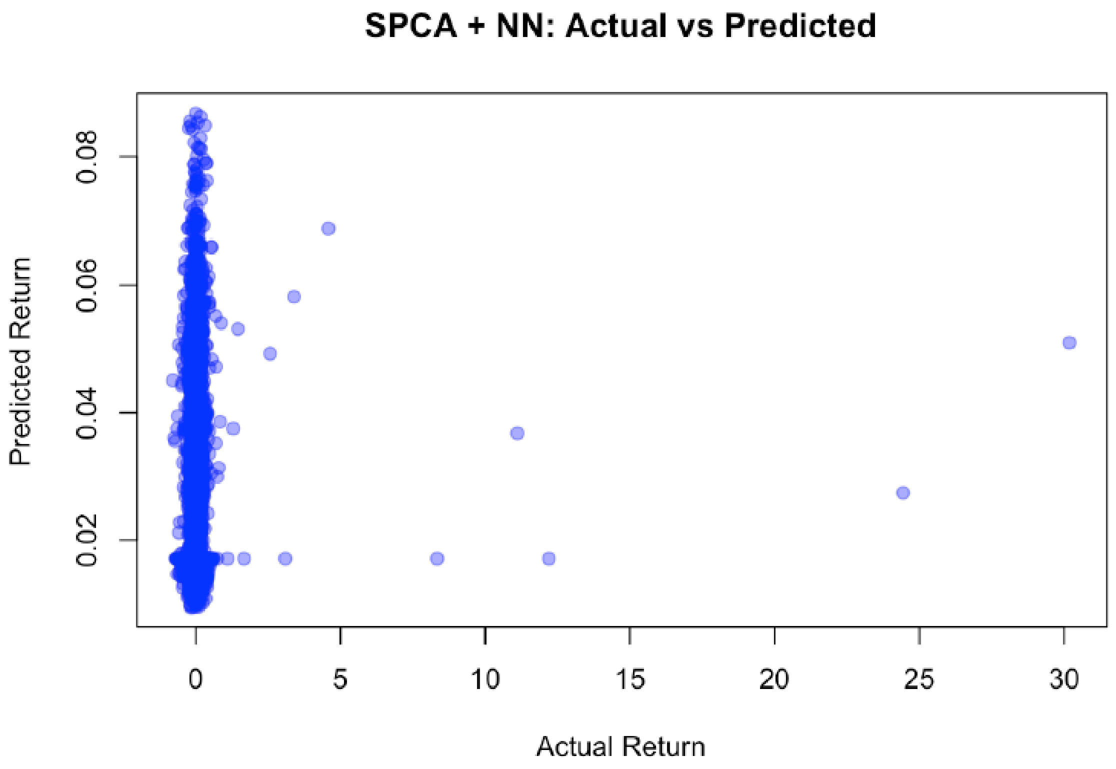


Figure 6 - SPCA+NN Actual vs Predicted Stock Returns

Portfolio Performance Comparison

Three portfolios have been constructed in total. To assess the practicality of machine learning models, two long-only portfolios were constructed on the predicted stock rankings from LASSO+NN and SPCA+NN models. Each month, the portfolio was formed by investing equally in the top 20% ranked stocks. The baseline portfolio with passive investing strategy is also constructed to hold S&P500 over the same period. To compare the performance of three portfolios, the average monthly returns, holding period returns, standard deviation of monthly returns and Sharpe ratios were calculated over the test period 2017-2018.

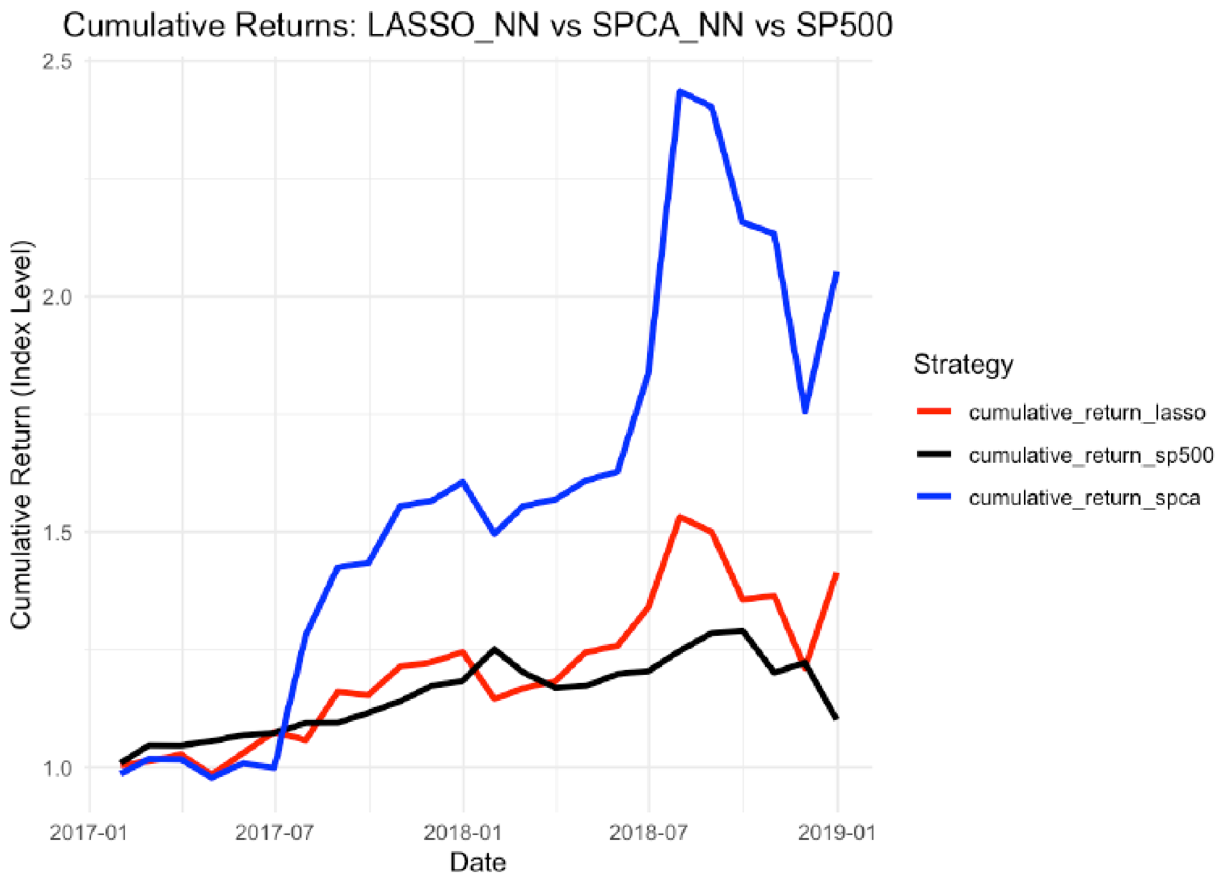
The SPCA+NN model achieves the highest performance across all metrics. It yields an average monthly return of 3.57%, a holding period return of 105% and a Sharpe ratio of 0.326. The LASSO+NN model generates a slightly lower average monthly return of 1.65%, a lower holding period return of 41.4% and a lower Sharpe ratio of 25.4%. Clearly, the SPCA+NN model outperforms LASSO+NN, which aligns with the previous analysis on training and test RMSE, where the SPCA+NN was deemed a slightly better model as it generalizes better on unseen data. In comparison against the passive strategy, the S&P500 investment achieves the lowest average monthly return of 0.46%, its holding period return is only 10.10%, significantly lower than the returns generated by the machine-learning-based models. Its Sharpe ratio of 0.135 is also significantly lower compared to the other two portfolios.

```
## # A tibble: 2 × 5
##   model  avg_monthly_return holding_period_return sd_monthly_return sharpe_ratio
##   <chr>               <dbl>                 <dbl>             <dbl>        <dbl>
## 1 LASSO…             0.0165                 0.414            0.0650        0.254
## 2 SPCA_…             0.0357                 1.05             0.109         0.326
```

Table 1 - LASSO+NN vs SPCA+NN Portfolio Performance

```
## # A tibble: 1 x 5
##   model avg_monthly_return holding_period_return sd_monthly_return sharpe_ratio
##   <chr>              <dbl>                 <dbl>             <dbl>        <dbl>
## 1 SP500            0.00458                 0.101            0.0339        0.135
```

Table 2 - Passive Investing on S&P500 Portfolio Performance



The cumulative return plot further visualizes the change in portfolio cumulative return over the test period (2017-2018) across three portfolios. Both machine-learning-based strategies significantly outperforms the S&P500 index, particularly during the bull market period in 2018. The blue line represents SPCA+NN strategy while the red line represents LASSO+NN strategy. SPCA-based strategy generates much higher return than the LASSO-based strategy, it also delivers better risk-adjusted performance as indicated by its relative lower monthly volatility and higher Sharpe ratio.

Figure 7 - Cumulative Returns Comparison between Three Portfolios

Related Work

“Projects 1 and 2 are highly related, therefore the reader is referred to our GPR #1 report for related work. Below is a quick summary of GRP #1 and what are the differences.
In GPR #1, we focused on the application of machine learning to factor investing, especially on SPCA, LASSO Regression, and neural networks as key tools for feature extraction and return prediction. In GPR #2, we successfully implemented both SPCA and LASSO, using each as a separate pipeline to generate inputs for our neural network model.
While SPCA is often applied to macroeconomic variables in literature, in our case, we applied it to firm-specific variables only, since our datasets contains solely of firm level financial and fundamental features. Separately we use LASSO regression to select most predictive factors directly. The features from each method were then used as inputs to feedforward neural networks, so we can compare predictive power and downstream portfolio performance.
Beyond what was covered in GPR #1, we did not do a long-short strategy (i.e.. Long top 10% and short bottom 10%), instead we constructed a long-only top 20% portfolio based on predicted returns and evaluated its performance using cumulative return and Sharpe ratio against the S&P 500 benchmark.

Conclusions

This research project explores the application machine learning techniques, especially Sparse Principal Component Analysis (SPCA) combined with Neural Networks and LASSO combined with Neural Networks, to stock return prediction and portfolio construction. The primary research objective is to determine whether machine-learning-based models can deliver superior investment performance relative to passive investing strategy, and whether feature generation via SPCA can improve predictability compared to dimensionality reduction via LASSO.

The portfolio performance results based on test data provide strong evidence that machine-learning-based models can enhance portfolio construction. Both LASSO-based and SPCA-based Neural Networks strategy significantly outperforms the passive S&P500 investment over 2017-2018 period in terms of holding period returns and Sharpe ratios. This suggests that properly regularize machine learning models can extract useful predictive signals to enhance investment outcomes. When comparing between machine-learning-based models, the higher holding period returns, and Sharpe ratio generated by SPCA-based strategy indicates that feature generation via SPCA offers advantages over direct feature selection using LASSO. The SPCA transformation can reduce noise and compress more information, which contributes to higher returns and lower volatility.

The use of Neural Networks in both strategies allow the non-linear complex relationships to be captured. Although the stock return predictions remain conservative, the model’s ability to rank stocks was sufficient to construct effective portfolios that can outperform passive investing strategy. Overall, combining dimensionality reduction or feature selections techniques with Neural Networks can improve stock selection and portfolio returns beyond passive benchmarks.

Appendix

See R code and Output Below

Untitled

2025-04-09

```
if(!require(readxl)){install.packages("readxl")}
library(readxl)
if(!require(dplyr)){install.packages("dplyr")}
library(dplyr)
if(!require(lubridate)){install.packages("lubridate")}
library(lubridate)
if(!require(tidyverse)){install.packages("tidyverse")}
library(tidyverse)
if (!require(glmnet)) install.packages("glmnet")
library(glmnet)
if (!require(keras)) install.packages("keras")
library(keras)
if (!require(elasticnet)) install.packages("elasticnet")
library(elasticnet)
if (!require(tidyr)) install.packages("tidyr")
library(tidyr)
if (!require(quantmod)) install.packages("quantmod")
library(quantmod)
if (!require(tidyquant)) install.packages("tidyquant")
library(tidyquant)
set.seed(123)
tensorflow::tf$random$set_seed(123)
```

```
# Use local drive address to load the data
load("data_ml.RData")
# preview first few rows of the dataset
head(data_ml, 6)
```

```
## # A tibble: 6 × 99
##   stock_id date       Advt_12M_Usd Advt_3M_Usd Advt_6M_Usd Asset_Turnover Bb_Yld
##      <int> <date>            <dbl>       <dbl>       <dbl>          <dbl>  <dbl>
## 1       13 2006-12-31         0.25        0.33        0.27           0.22   0.33
## 2       13 2007-01-31         0.25        0.32        0.28           0.22   0.4
## 3       13 2007-02-28         0.26        0.3         0.3            0.22   0.15
## 4       17 2015-03-31         0.73        0.64        0.7            0.4    0.47
## 5       17 2015-04-30         0.72        0.62        0.66           0.4    0.46
## 6       17 2015-05-31         0.71        0.63        0.64           0.4    0.47
## # ℹ 92 more variables: Bv <dbl>, Capex_Ps_Cf <dbl>, Capex_Sales <dbl>,
## #   Cash_Div_Cf <dbl>, Cash_Per_Share <dbl>, Cf_Sales <dbl>, Debtequity <dbl>,
## #   Div_Yld <dbl>, Dps <dbl>, Ebit_Bv <dbl>, Ebit_Noa <dbl>, Ebit_Oa <dbl>,
## #   Ebit_Ta <dbl>, Ebitda_Margin <dbl>, Eps <dbl>, Eps_Basic <dbl>,
## #   Eps_Basic_Gr <dbl>, Eps_Contin_Oper <dbl>, Eps_Dil <dbl>, Ev <dbl>,
## #   Ev_Ebitda <dbl>, Fa_Ci <dbl>, Fcf <dbl>, Fcf_Bv <dbl>, Fcf_Ce <dbl>,
## #   Fcf_Margin <dbl>, Fcf_Noa <dbl>, Fcf_Oa <dbl>, Fcf_Ta <dbl>, …
```

```
# Download SP500 index (^GSPC) from Yahoo Finance
# Get daily data first
sp500_data <- tq_get("^GSPC",
                     from = "2016-12-31",
                     to = "2018-12-31",
                     get = "stock.prices")

# Calculate monthly simple returns (not log returns for consistency)
sp500_monthly_returns <- sp500_data %>%
  tq_transmute(
    select = adjusted,
    mutate_fun = periodReturn,
    period = "monthly",
    type = "arithmetic"  # simple returns
  ) %>%
  rename(sp500_return = monthly.returns)
```

```
# Clean and sort the dataset
target_leakage <- c("R3M_Usd", "R6M_Usd", "R12M_Usd")
data_ml <- data_ml %>%
  distinct() %>% #remove duplicates
  filter(date > "1999-12-31",         # Keep the date with sufficient data points
         date < "2019-01-01") %>%
  select(-all_of(target_leakage)) %>% # remove predictors that may cause target leakage
  arrange(stock_id, date)             # Order the data
```

```
data_ml <- data_ml %>%
  mutate(target_return = R1M_Usd) %>%
  filter(!is.na(target_return))  # Remove rows without a future return

# verify predictors are all scaled (standardized)
col_mean <- apply(data_ml[, 3:ncol(data_ml)], 2, mean)
col_sd <- apply(data_ml[, 3:ncol(data_ml)], 2, sd)

standardization_check <- data.frame(Column = colnames(data_ml[, 3:ncol(data_ml)]),
                                    mean = col_mean,
                                    sd = col_sd)

mean_range <- c(min(standardization_check$mean), max(standardization_check$mean))
sd_range <- c(min(standardization_check$sd), max(standardization_check$sd))

cat('Mean range:', mean_range[1], '-', mean_range[2], '\n')
```

```
## Mean range: 0.01273207 - 0.5056039
```

```
cat('Standard Deviation range:', sd_range[1], '-', sd_range[2], '\n')
```

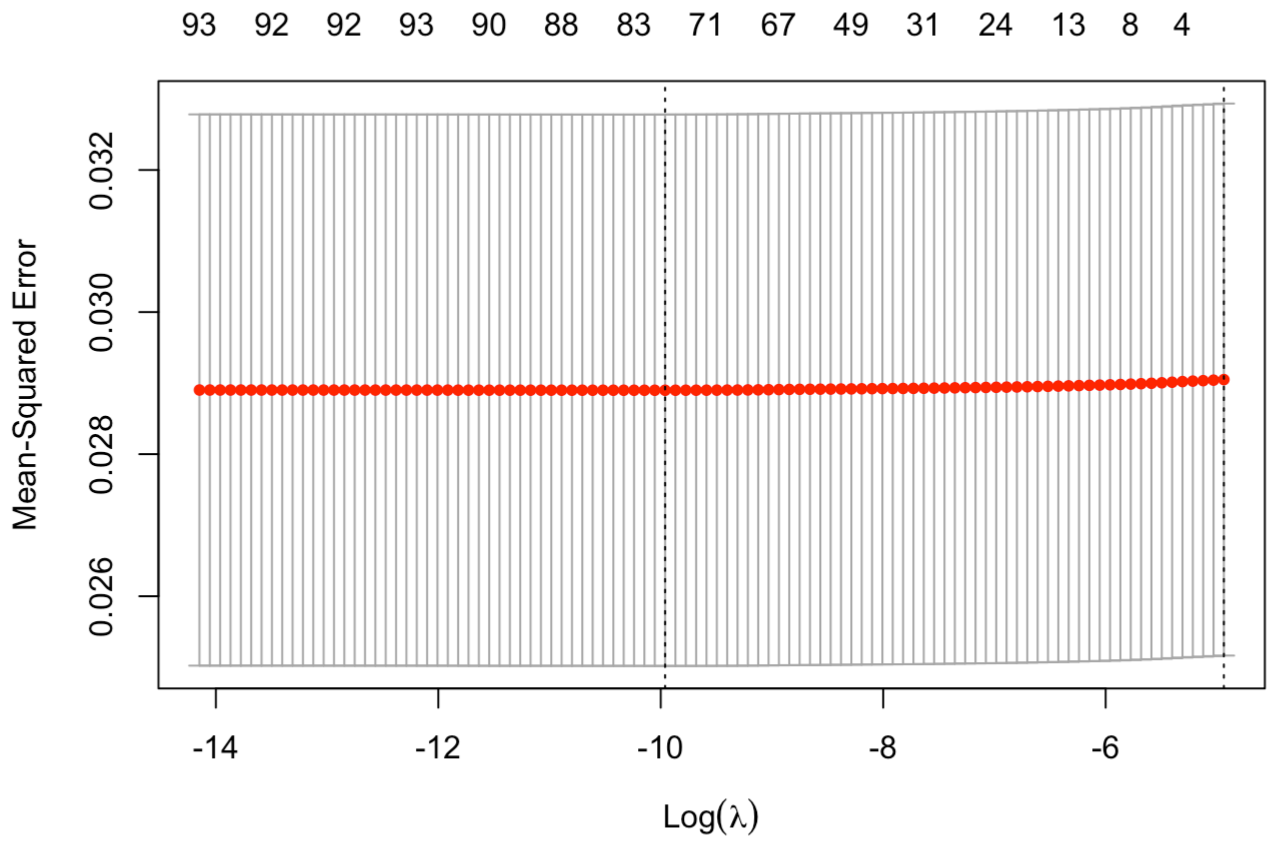
```
## Standard Deviation range: 0.1764311 - 0.2890281
```

```r
# standardize the predictors
data_ml <- data_ml %>% mutate(across(
  .cols = -all_of(c('stock_id', 'date', 'R1M_Usd', 'target_return')),
  .fns = ~scale(.)[,1]
))
```

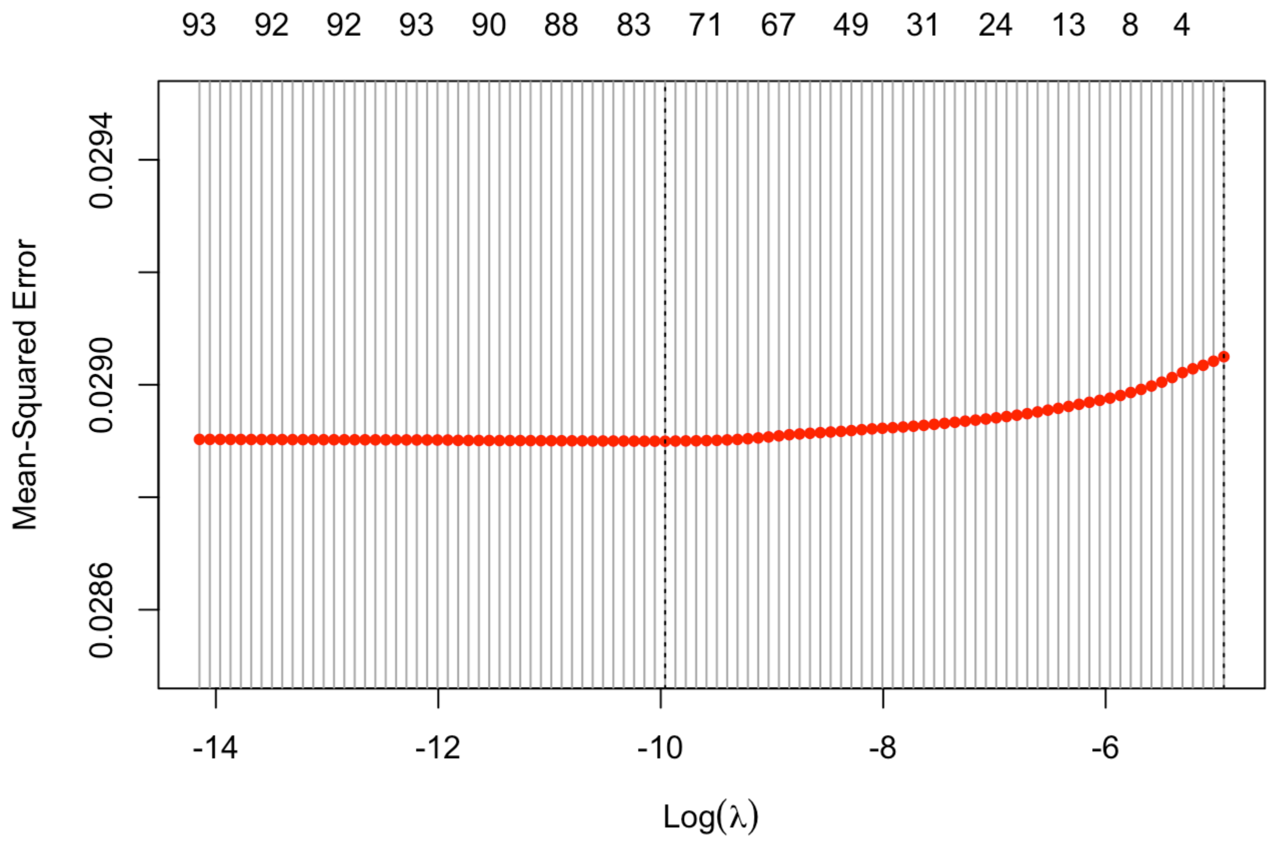
```r
# Split into training, validation, and test sets
training_set <- filter(data_ml, date < as.Date("2014-01-15"))
validation_set <- filter(data_ml, (date >= as.Date("2014-01-15")
                                   & date < as.Date("2017-01-15")))
testing_set <- filter(data_ml, date >= as.Date("2017-01-15"))
```

```r
# Define features to be used for LASSO and NN
features <- training_set %>%
  select(-stock_id, -date, -R1M_Usd, -target_return) %>%
  colnames()

# Create matrix inputs for glmnet
X_train <- as.matrix(training_set[, features])
y_train <- training_set$target_return

X_val <- as.matrix(validation_set[, features])
y_val <- validation_set$target_return

X_test <- as.matrix(testing_set[, features])
y_test <- testing_set$target_return
```

```r
# Train LASSO model with cross-validation
cv_lasso <- cv.glmnet(X_train, y_train, alpha = 1, nfolds = 10)

# plot CV error vs. lambda
plot(cv_lasso)
```



```
plot(cv_lasso, ylim=c(0.0285, 0.0295))
```



```
# Choose lambda with lowest CV error
best_lambda <- cv_lasso$lambda.min
cat("Best lambda:", best_lambda, "\n")
```

```
## Best lambda: 4.720529e-05
```

```
# Extract coefficients at best lambda
lasso_coef <- coef(cv_lasso, s = best_lambda)
selected_features <- rownames(lasso_coef)[which(lasso_coef[, 1] != 0)]
selected_features <- setdiff(selected_features, "(Intercept)")  # remove intercept
lasso_coef <- as.matrix(lasso_coef) %>%
  as.data.frame() %>%
  slice(-1) %>%
  filter(s1 !=0) %>%
  arrange(desc(abs(s1)))
lasso_coef
```

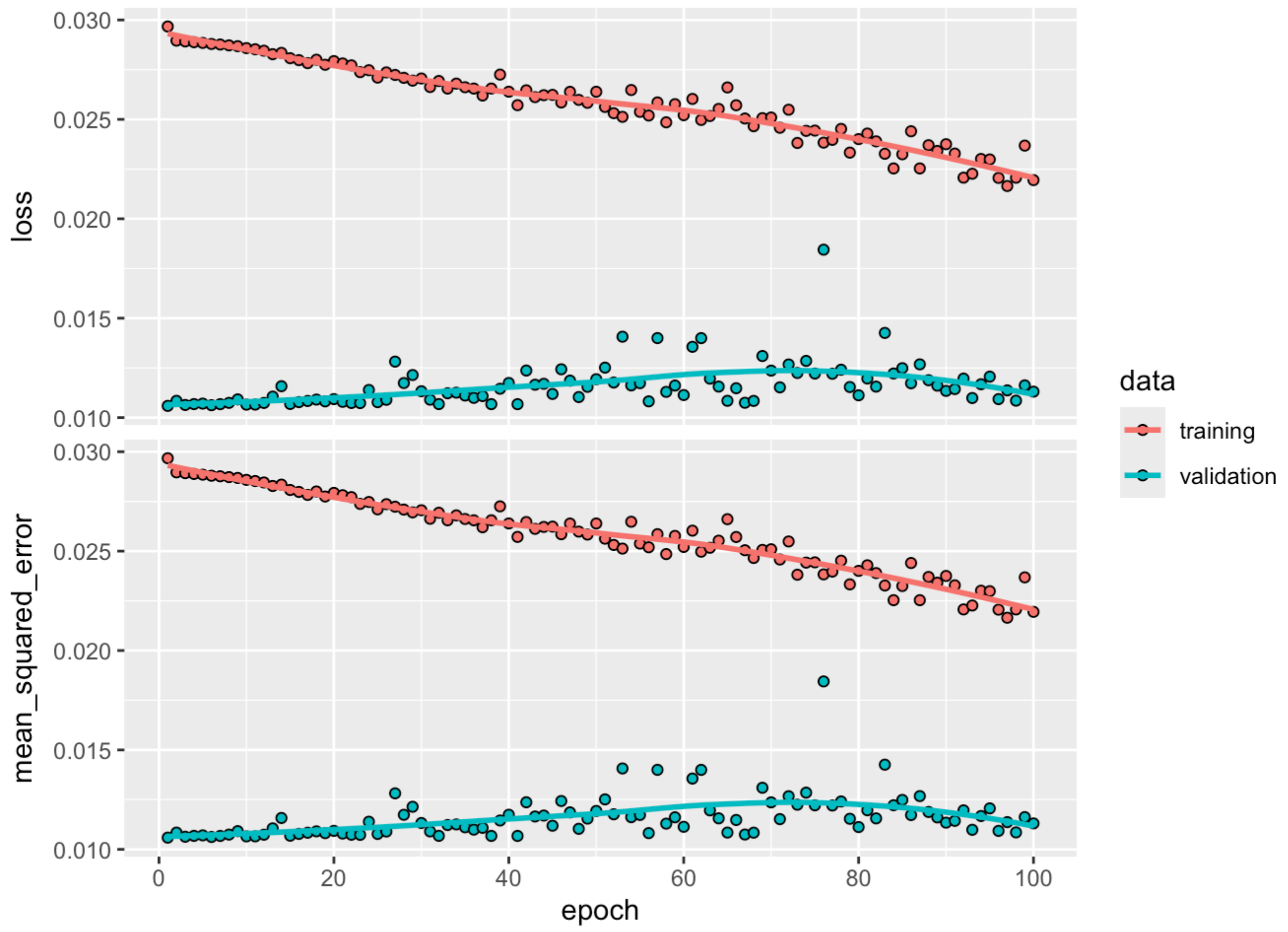
```
##                                           s1
## Mkt_Cap_3M_Usd                 -3.629189e-02
## Mkt_Cap_12M_Usd                 2.723486e-02
## Fcf_Bv                          5.963966e-03
## Ocf_Ta                          5.231126e-03
## Mom_5M_Usd                     -5.048371e-03
## Pb                             -4.829150e-03
## Fcf_Yld                        -4.545451e-03
## Ocf_Bv                         -4.478071e-03
## Mom_Sharp_11M_Usd               4.362510e-03
## Ni                              4.138691e-03
## Rev                             4.101435e-03
## Mom_11M_Usd                    -3.778312e-03
## Mom_Sharp_5M_Usd                3.566957e-03
## Ni_Oa                          -3.526356e-03
## Sales_Ps                       -3.470986e-03
## Ebitda_Margin                   3.230367e-03
## Fcf                             3.210102e-03
## Ocf_Margin                     -3.144972e-03
## Total_Debt                      2.865797e-03
## Vol1Y_Usd                       2.705398e-03
## Debtequity                     -2.623354e-03
## Vol3Y_Usd                       2.594227e-03
## Pe                             -2.570003e-03
## Mkt_Cap_6M_Usd                 -2.483954e-03
## Ni_Avail_Margin                 2.454994e-03
## Bv                             -2.236407e-03
## Roce                            2.165793e-03
## Eps                            -2.144026e-03
## Asset_Turnover                 -2.106678e-03
## Cash_Per_Share                 -2.101890e-03
## Ocf_Oa                          2.063179e-03
## Net_Margin                     -2.037736e-03
## Share_Turn_3M                   2.025437e-03
## Total_Liabilities_Total_Assets  2.017764e-03
## Fcf_Tbv                         1.919880e-03
## Share_Turn_12M                 -1.891544e-03
## Oper_Ps_Net_Cf                  1.882790e-03
## Ocf_Tbv                        -1.750312e-03
## Fa_Ci                           1.748320e-03
## Ni_Toa                          1.701317e-03
## Eps_Contin_Oper                -1.660257e-03
## Cf_Sales                       -1.589986e-03
## Ev_Ebitda                       1.577615e-03
## Ocf                             1.543912e-03
## Ebit_Noa                       -1.507896e-03
## Advt_3M_Usd                    -1.475341e-03
## Cash_Div_Cf                    -1.460924e-03
## Dps                             1.444972e-03
## Recurring_Earning_Total_Assets -1.292998e-03
## Div_Yld                        -1.255963e-03
## Capex_Sales                    -1.211795e-03
## Tev_Less_Mktcap                 1.183148e-03
## Roa                            -1.149997e-03
## Fcf_Ce                          1.121742e-03
## Capex_Ps_Cf                    -1.044732e-03
## Fcf_Margin                     -1.019915e-03
## Total_Debt_Capital              8.994458e-04
## Op_Margin                      -8.774057e-04
## Fcf_Toa                        -8.643440e-04
## Ebit_Oa                        -8.529555e-04
## Interest_Expense                8.319396e-04
## Return_On_Capital               8.155614e-04
## Ebit_Ta                        -8.129246e-04
## Fcf_Oa                         -8.058481e-04
## Net_Debt                       -7.743785e-04
## Roc                             7.439507e-04
## Int_Rev                        -6.715502e-04
## Ebit_Bv                        -6.461071e-04
## Ptx_Mgn                         6.414147e-04
## Bb_Yld                          6.220725e-04
## Tot_Debt_Rev                   -6.179719e-04
## Eps_Basic_Gr                   -4.634775e-04
## Fcf_Ta                         -4.352807e-04
## Fcf_Noa                         2.200777e-04
## Eps_Dil                         1.469795e-04
## Ocf_Ce                          1.099358e-04
## Oa                             -8.445767e-05
## Advt_12M_Usd                    2.002129e-05
## Total_Capital                   1.956835e-05
## Free_Ps_Cf                     -7.786797e-06
## Eps_Basic                       4.753347e-06
```

```
# Prepare training, validation, and test sets using only selected LASSO features
X_train_nn_lasso <- as.matrix(training_set[, selected_features])
X_val_nn_lasso   <- as.matrix(validation_set[, selected_features])
X_test_nn_lasso  <- as.matrix(testing_set[, selected_features])

y_train_nn_lasso <- training_set$target_return
y_val_nn_lasso   <- validation_set$target_return
y_test_nn_lasso  <- testing_set$target_return
```

```
# Define a grid of hyperparameters
units1_list <- c(16, 32, 64)
units2_list <- c(8, 16, 32)
learning_rates <- c(0.001, 0.0005)
batch_sizes <- c(64, 128)

results <- data.frame()

for (units1 in units1_list) {
  for (units2 in units2_list) {
    for (lr in learning_rates) {
      for (bs in batch_sizes) {

        # Build model
        # Starts a new model using a sequential stack of layers.
        # Layer 1.  x neurons in the first hidden layer.
        #     Applies ReLU activation (max(0, x)), good for non-linearity.
        #     Number of input features (i.e., number of selected LASSO predictors).
        # Layer 2.  x neurons, again with ReLU.
        # Layer 3.  1 output neuron: predicting a return.
        model <- keras_model_sequential() %>%
          layer_dense(units = units1, activation = "relu", input_shape = ncol(X_train_nn
_lasso)) %>%
          layer_dense(units = units2, activation = "relu") %>%
          layer_dense(units = 1)

        # Compile model
        # Mean Squared Error is the objective to minimize (standard for regression).
        # Uses the Adam optimizer, which is adaptive and works well with most problems.
        # Also tracks MSE while training, for reporting
        # Learning Rate x
        model %>% compile(
          loss = "mse",
          optimizer = optimizer_adam(learning_rate = lr),
          metrics = list("mean_squared_error")
        )

        # Train model
        # Train the model for 100 full passes through the data.
        # Use mini-batches of x rows for updates (tradeoff between speed and stability).
        # Monitors performance on validation set at the end of each epoch.
        history <- model %>% fit(
          x = X_train_nn_lasso,
          y = y_train_nn_lasso,
          epochs = 100,
          batch_size = bs,
          validation_data = list(X_val_nn_lasso, y_val_nn_lasso),
          verbose = 0
        )

        # Get final validation MSE
        val_mse <- tail(history$metrics$val_mean_squared_error, 1)
        # Record result
        results <- rbind(results, data.frame(
          units1 = units1,
          units2 = units2,
          learning_rate = lr,
          batch_size = bs,
          val_mse = val_mse
        ))
      }
    }
  }
}

# Find the best hyperparameters
best_result <- results[which.min(results$val_mse), ]
print(best_result)
```

```
##   units1 units2 learning_rate batch_size   val_mse
## 2     16      8         0.001        128 0.0106187
```

```r
# Build model
# Starts a new model using a sequential stack of layers.
# Layer 1.  x neurons in the first hidden layer.
#     Applies ReLU activation (max(0, x)), good for non-linearity.
#     Number of input features (i.e., number of selected LASSO predictors).
# Layer 2.  x neurons, again with ReLU.
# Layer 3.  1 output neuron: predicting a return.
model <- keras_model_sequential() %>%
  layer_dense(units = best_result$units1, activation = "relu",
              input_shape = ncol(X_train_nn_lasso)) %>%
  layer_dense(units = best_result$units2, activation = "relu") %>%
  layer_dense(units = 1)  # output layer

# Compile model
# Mean Squared Error is the objective to minimize (standard for regression).
# Uses the Adam optimizer, which is adaptive and works well with most problems.
# Also tracks MSE while training, for reporting
# Learning Rate x
model %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(learning_rate = best_result$learning_rate),
  metrics = list("mean_squared_error")
)

# Train model
# Train the model for 100 full passes through the data.
# Use mini-batches of x rows for updates (tradeoff between speed and stability).
# Monitors performance on validation set at the end of each epoch.
history <- model %>% fit(
  x = X_train_nn_lasso,
  y = y_train_nn_lasso,
  epochs = 100,
  batch_size = best_result$batch_size,
  validation_data = list(X_val_nn_lasso, y_val_nn_lasso),
  verbose = 0
)
plot(history)
```



```
#title(main = "LASSO-Selected Features Neural Network")
```

```
# Predict and calculate RMSE
pred_nn_lasso_train <- model %>% predict(X_train_nn_lasso)
```

```
## 6192/6192 - 1s - 956ms/epoch - 154us/step
```

```
pred_nn_lasso_test <- model %>% predict(X_test_nn_lasso)
```

```
## 853/853 - 0s - 134ms/epoch - 158us/step
```

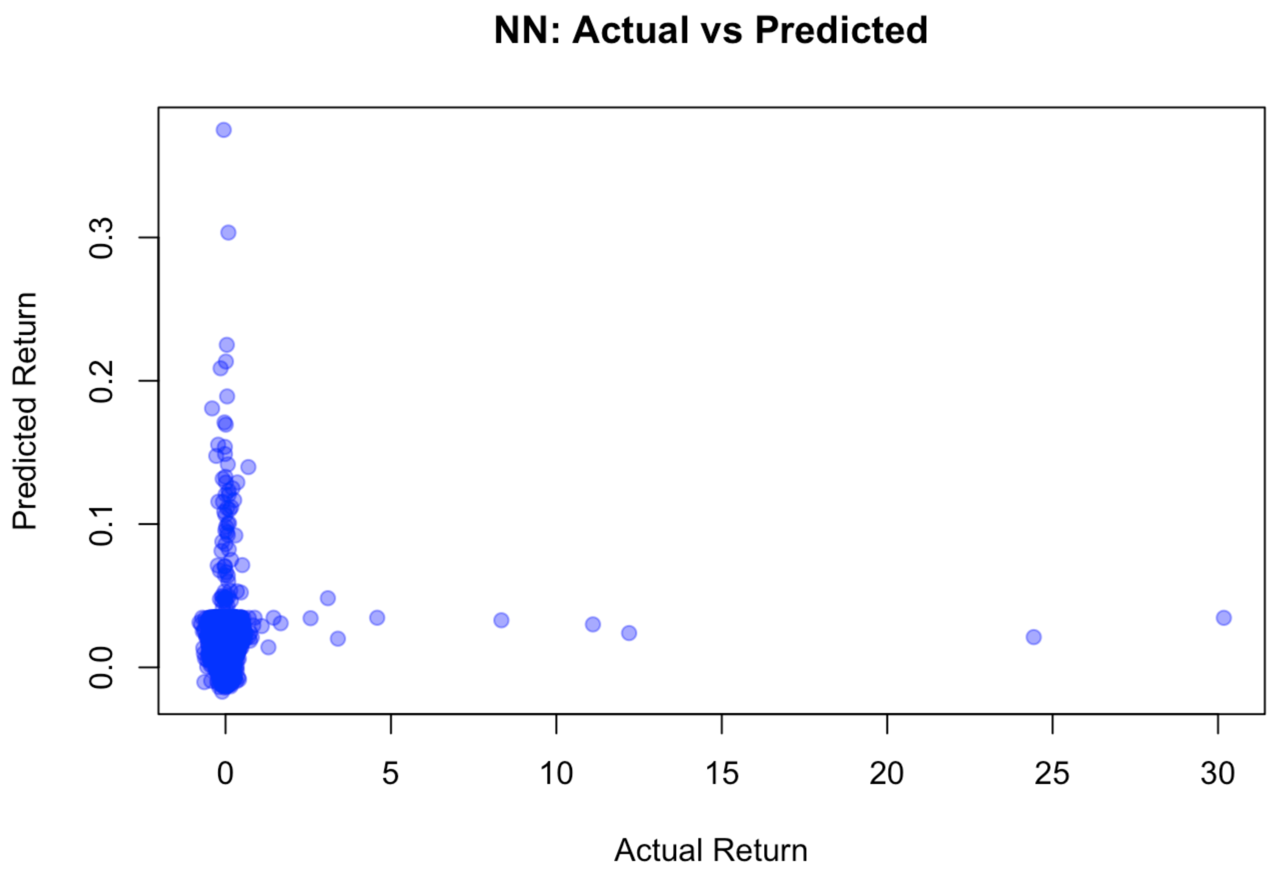
```
rmse <- function(actual, predicted) {
  sqrt(mean((actual - predicted)^2))
}

cat("LASSO + NN Train RMSE:", rmse(y_train_nn_lasso, pred_nn_lasso_train), "\n")
```

```
## LASSO + NN Train RMSE: 0.1466643
```

```
cat("LASSO + NN Test RMSE:", rmse(y_test_nn_lasso, pred_nn_lasso_test), "\n")
```

```
## LASSO + NN Test RMSE: 0.2803473
```

```
plot(y_test_nn_lasso, pred_nn_lasso_test,
     xlab = "Actual Return",
     ylab = "Predicted Return",
     main = "NN: Actual vs Predicted",
     pch = 19, col = rgb(0,0,1,0.4))
```



```
# SPCA transformed matrices for training, validation and test
X_train_spca <- as.matrix(training_set[, features])
X_val_spca <- as.matrix(validation_set[, features])
X_test_spca <- as.matrix(testing_set[, features])

y_train_nn_spca <- training_set$target_return
y_val_nn_spca  <- validation_set$target_return
y_test_nn_spca <- testing_set$target_return
```

```
# Run SPCA for dimensionality reduction
spca_result <- suppressMessages(spca(X_train_spca,
                    K = 5,                      # Number of components
                    type = "predictor",
                    sparse = "penalty",
                    para = rep(0.4, 5),         # Sparsity level per component
                    trace = FALSE))
```

```
## You may wish to restart and use a more efficient way
## let the argument x be the sample covariance/correlation matrix and set type=Gram
```

```
print(spca_result$loadings)
```

```
##                                           PC1          PC2          PC3
## Advt_12M_Usd                     0.1443918664  0.074803695  0.197313054
## Advt_3M_Usd                      0.1453647325  0.073585996  0.195172438
## Advt_6M_Usd                      0.1453854598  0.074049701  0.196310695
## Asset_Turnover                   0.0070275493 -0.116467774  0.188712406
## Bb_Yld                           0.0627610363 -0.014908479  0.011521453
## Bv                               0.1414199638  0.132924597  0.097593411
## Capex_Ps_Cf                      0.0362685657  0.100756628  0.077701138
## Capex_Sales                     -0.0048421910  0.057384712  0.029679002
## Cash_Div_Cf                      0.0487712420  0.067106725 -0.143976401
## Cash_Per_Share                   0.1163549738 -0.065378501 -0.035186736
## Cf_Sales                         0.0861390004  0.007892096 -0.149661562
## Debtequity                       0.0188107336  0.177876012 -0.084763453
## Div_Yld                          0.0484760464  0.084699955 -0.139216332
## Dps                              0.0812289456  0.094046419 -0.128904151
## Ebit_Bv                          0.1102990626  0.001504903 -0.141987633
## Ebit_Noa                         0.1003472917 -0.081890134 -0.127323472
## Ebit_Oa                          0.1045652317 -0.084249721 -0.147197825
## Ebit_Ta                          0.1088235140 -0.070489933 -0.150307267
## Ebitda_Margin                    0.0829720224  0.042283296 -0.213115789
## Eps                              0.1493786050  0.014800086 -0.076313400
## Eps_Basic                        0.1494401192  0.015135417 -0.075270599
## Eps_Basic_Gr                     0.0554218635 -0.051140637 -0.009316707
## Eps_Contin_Oper                  0.1455306168  0.014134570 -0.072644232
## Eps_Dil                          0.1498037171  0.016052318 -0.076747672
## Ev                               0.1638489244  0.130455993  0.103878568
## Ev_Ebitda                        0.0092733667 -0.033197012  0.109561945
## Fa_Ci                            0.0123101004  0.073048750  0.063640665
## Fcf                              0.1701309168 -0.010708859  0.025548296
## Fcf_Bv                           0.1252168584 -0.099764429 -0.040543902
## Fcf_Ce                           0.0996747152 -0.091146248 -0.066451082
## Fcf_Margin                       0.1085252969 -0.079271845 -0.134407612
## Fcf_Noa                          0.1110305016 -0.155732269 -0.011485390
## Fcf_Oa                           0.1122866028 -0.168314556  0.026548882
## Fcf_Ta                           0.1183780406 -0.162364489  0.022322940
## Fcf_Tbv                          0.0631456714 -0.072839351 -0.031133502
## Fcf_Toa                          0.0892001696 -0.134482432  0.014514732
## Fcf_Yld                          0.0863521825 -0.057524016 -0.093387031
## Free_Ps_Cf                       0.1163430329 -0.065367112 -0.035181874
## Int_Rev                          0.0199472236  0.015701003 -0.030351640
## Interest_Expense                 0.0897631078  0.183329132  0.046416501
## Mkt_Cap_12M_Usd                  0.1689828873  0.091612461  0.134221385
## Mkt_Cap_3M_Usd                   0.1694629889  0.089082165  0.132917089
## Mkt_Cap_6M_Usd                   0.1696287311  0.089882924  0.133534330
## Mom_11M_Usd                      0.0124077208 -0.018709112 -0.009759618
## Mom_5M_Usd                       0.0014306416 -0.013135485 -0.003804307
## Mom_Sharp_11M_Usd                0.0279070289 -0.007758183 -0.029488238
## Mom_Sharp_5M_Usd                 0.0163428548 -0.004230579 -0.021344923
## Nd_Ebitda                       -0.0182194135  0.167234858 -0.042196440
## Net_Debt                         0.0599277994  0.196624599 -0.033873660
## Net_Debt_Cf                      0.0008057218  0.038063636 -0.022378451
## Net_Margin                       0.1218463339 -0.045279817 -0.175551854
## Netdebtyield                       0.0010257760  0.039847310 -0.023751457
## Ni                                 0.1903191379  0.043086860  0.033358177
## Ni_Avail_Margin                    0.1171068234 -0.047827547 -0.165733748
## Ni_Oa                              0.1148311590 -0.161202078  0.045779400
## Ni_Toa                             0.0947407177 -0.130068901  0.021172509
## Noa                                0.1298282112  0.185640487  0.050344208
## Oa                                 0.1322683489  0.184173942  0.006830147
## Ocf                                0.1818058490  0.077256196  0.085913672
## Ocf_Bv                             0.1295613833 -0.057242258  0.024689021
## Ocf_Ce                             0.0989876217 -0.086462335 -0.022435992
## Ocf_Margin                         0.0973819349 -0.012772864 -0.141838874
## Ocf_Noa                            0.1067277896 -0.162566006  0.061387673
## Ocf_Oa                             0.0999405882 -0.170090288  0.097646731
## Ocf_Ta                             0.1095853622 -0.157464218  0.092160535
## Ocf_Tbv                            0.0661418375 -0.054908577  0.004449299
## Ocf_Toa                            0.0815152849 -0.136552302  0.061356923
## Op_Margin                          0.1212653819 -0.021751076 -0.170652221
## Op_Prt_Margin                      0.0500281001 -0.039210232 -0.026410298
## Oper_Ps_Net_Cf                     0.1320979793  0.052969769 -0.033168664
## Pb                                 0.0752847218 -0.085722653  0.080225112
## Pe                                -0.0420691410 -0.023374809  0.084593838
## Ptx_Mgn                            0.1242740054 -0.049024495 -0.172843239
## Recurring_Earning_Total_Assets     0.0628972520 -0.110639684  0.064662067
## Return_On_Capital                  0.1270517092 -0.134797174  0.038832862
## Rev                                0.1371442347  0.100146070  0.170131734
## Roa                                0.1202931797 -0.151958768  0.044026168
## Roc                                0.1270491881 -0.134818268  0.038839858
## Roce                               0.1242094958 -0.037302762  0.033975139
## Roe                                0.1469162770 -0.089396216 -0.023272007
## Sales_Ps                           0.0512094861  0.062660717  0.082281001
## Share_Turn_12M                     0.0076245886 -0.004981105  0.223696895
## Share_Turn_3M                      0.0107027229 -0.003099436  0.219080626
## Share_Turn_6M                      0.0096663381 -0.003887778  0.222076348
## Ta                                 0.1356201154  0.179073015  0.018054573
## Tev_Less_Mktcap                    0.0589988590  0.169754458 -0.025170318
## Tot_Debt_Rev                       0.0079835132  0.177302184 -0.144321024
## Total_Capital                      0.1375999780  0.178208180  0.069858056
## Total_Debt                         0.1020459739  0.207041611  0.012017399
## Total_Debt_Capital                 0.0169966187  0.173009910 -0.079271603
## Total_Liabilities_Total_Assets     0.0290710506  0.147287785 -0.119841024
## Vol1Y_Usd                         -0.0922334444 -0.055806512  0.129612383
## Vol3Y_Usd                         -0.0896649093 -0.058425129  0.136127322
##                                             PC4           PC5
## Advt_12M_Usd                       0.0018691304 -0.1075240770
## Advt_3M_Usd                        0.0019894222 -0.1090588524
## Advt_6M_Usd                        0.0017220989 -0.1091094580
## Asset_Turnover                    -0.0807416651  0.2187236257
## Bb_Yld                            -0.0002789474  0.0337026682
## Bv                                 0.0202656943 -0.0265724126
## Capex_Ps_Cf                       -0.1504790716  0.0794760220
## Capex_Sales                       -0.1382373371 -0.1949035650
## Cash_Div_Cf                       -0.0158322271 -0.0036905277
## Cash_Per_Share                0.2149921544  0.1497583904
## Cf_Sales                      0.0080620151 -0.2712193648
## Debtequity                    0.0516171391  0.0177911583
## Div_Yld                      -0.0159854699  0.0364210823
## Dps                          -0.0361133113  0.0469543646
## Ebit_Bv                      -0.0105149719  0.0923948202
## Ebit_Noa                     -0.0189197880  0.0545740124
## Ebit_Oa                      -0.0322310103  0.0322370551
## Ebit_Ta                      -0.0407047109  0.0509473723
## Ebitda_Margin                 0.0152448235 -0.1746970826
## Eps                          -0.1736210643  0.1432612092
## Eps_Basic                    -0.1719276349  0.1435808081
## Eps_Basic_Gr                 -0.1127970135  0.0026341340
## Eps_Contin_Oper              -0.1689683870  0.1424406862
## Eps_Dil                      -0.1722364367  0.1437921663
## Ev                            0.0117561599 -0.0722101773
## Ev_Ebitda                    -0.0312213939 -0.1728267632
## Fa_Ci                        -0.1065157711  0.0485329914
## Fcf                           0.1881395494  0.0502074441
## Fcf_Bv                        0.2221580584  0.0670444506
## Fcf_Ce                        0.1737042823 -0.0591950693
## Fcf_Margin                    0.2289880593 -0.0842778978
## Fcf_Noa                       0.1813715515  0.0373557898
## Fcf_Oa                        0.1658283543  0.0098156226
## Fcf_Ta                        0.1690907610  0.0235883534
## Fcf_Tbv                       0.1026586887 -0.0009732250
## Fcf_Toa                       0.1069857367 -0.0528281909
## Fcf_Yld                       0.2747876640  0.1324715681
## Free_Ps_Cf                    0.2148773513  0.1497529888
## Int_Rev                       0.0560438032 -0.0974830242
## Interest_Expense              0.0479204637  0.0090661163
## Mkt_Cap_12M_Usd              -0.0071728585 -0.0793161560
## Mkt_Cap_3M_Usd               -0.0064710607 -0.0817917793
## Mkt_Cap_6M_Usd               -0.0070571824 -0.0814351244
## Mom_11M_Usd                   0.0053970310 -0.0142456394
## Mom_5M_Usd                    0.0152312193 -0.0033259588
## Mom_Sharp_11M_Usd            -0.0031392832 -0.0155609850
## Mom_Sharp_5M_Usd              0.0069114396 -0.0046753246
## Nd_Ebitda                     0.0208535625  0.0216474014
## Net_Debt                      0.0243749749  0.0432810815
## Net_Debt_Cf                  -0.1088958467 -0.0553526208
## Net_Margin                   -0.1262617936 -0.1849044247
## Netdebtyield                 -0.1145235696 -0.0609111179
## Ni                           -0.1082443889  0.0182093648
## Ni_Avail_Margin              -0.1244354489 -0.1696689571
## Ni_Oa                        -0.1763604106 -0.0165566137
## Ni_Toa                       -0.1289435368 -0.0571699759
## Noa                           0.0323470062 -0.0012094250
## Oa                            0.0529746455  0.0201052957
## Ocf                           0.0728056732 -0.0024649718
## Ocf_Bv                        0.0865001338  0.0468217863
## Ocf_Ce                        0.0820281274 -0.0906416273
## Ocf_Margin                        0.0966603698 -0.2411669222
## Ocf_Noa                           0.0538191123  0.0125392299
## Ocf_Oa                            0.0200600588 -0.0414280182
## Ocf_Ta                            0.0144942133 -0.0218961369
## Ocf_Tbv                           0.0355202489  0.0003532267
## Ocf_Toa                           0.0216164300 -0.0820527058
## Op_Margin                        -0.0878388276 -0.1799745444
## Op_Prt_Margin                    -0.1022933249  0.0067622669
## Oper_Ps_Net_Cf                    0.0533033668  0.1359404007
## Pb                               -0.0618417929 -0.1260436279
## Pe                                0.0871731110 -0.1857895071
## Ptx_Mgn                          -0.1260240997 -0.1775555055
## Recurring_Earning_Total_Assets -0.1092690072  0.1132502137
## Return_On_Capital                -0.1390604769  0.0734466306
## Rev                               0.0048295306  0.1416106428
## Roa                              -0.1900218518 -0.0067675813
## Roc                              -0.1391459537  0.0734708099
## Roce                             -0.0523463752  0.0283726998
## Roe                              -0.1680083028  0.0419337564
## Sales_Ps                         -0.0407757561  0.3379715488
## Share_Turn_12M                    0.0184418138 -0.1146002644
## Share_Turn_3M                     0.0158386235 -0.1135380134
## Share_Turn_6M                     0.0166336745 -0.1147678473
## Ta                                0.0571156405  0.0054631763
## Tev_Less_Mktcap                   0.0310823977  0.0126227771
## Tot_Debt_Rev                      0.0541416893 -0.1103541994
## Total_Capital                     0.0404375949 -0.0236776514
## Total_Debt                        0.0474198256  0.0121654649
## Total_Debt_Capital                0.0507363994  0.0189526800
## Total_Liabilities_Total_Assets  0.0806651577  0.0985842225
## Vol1Y_Usd                         0.0440568506 -0.0358280218
## Vol3Y_Usd                         0.0515387797 -0.0495200248
```

```
# loadings < 0.05 are considered close to zero and insignficant
percentage <- mean(spca_result$loadings <0.05)*100
cat('Percentage of close to zero loadings (< 0.05):', percentage, "\n")
```

```
## Percentage of close to zero loadings (< 0.05): 62.15054
```

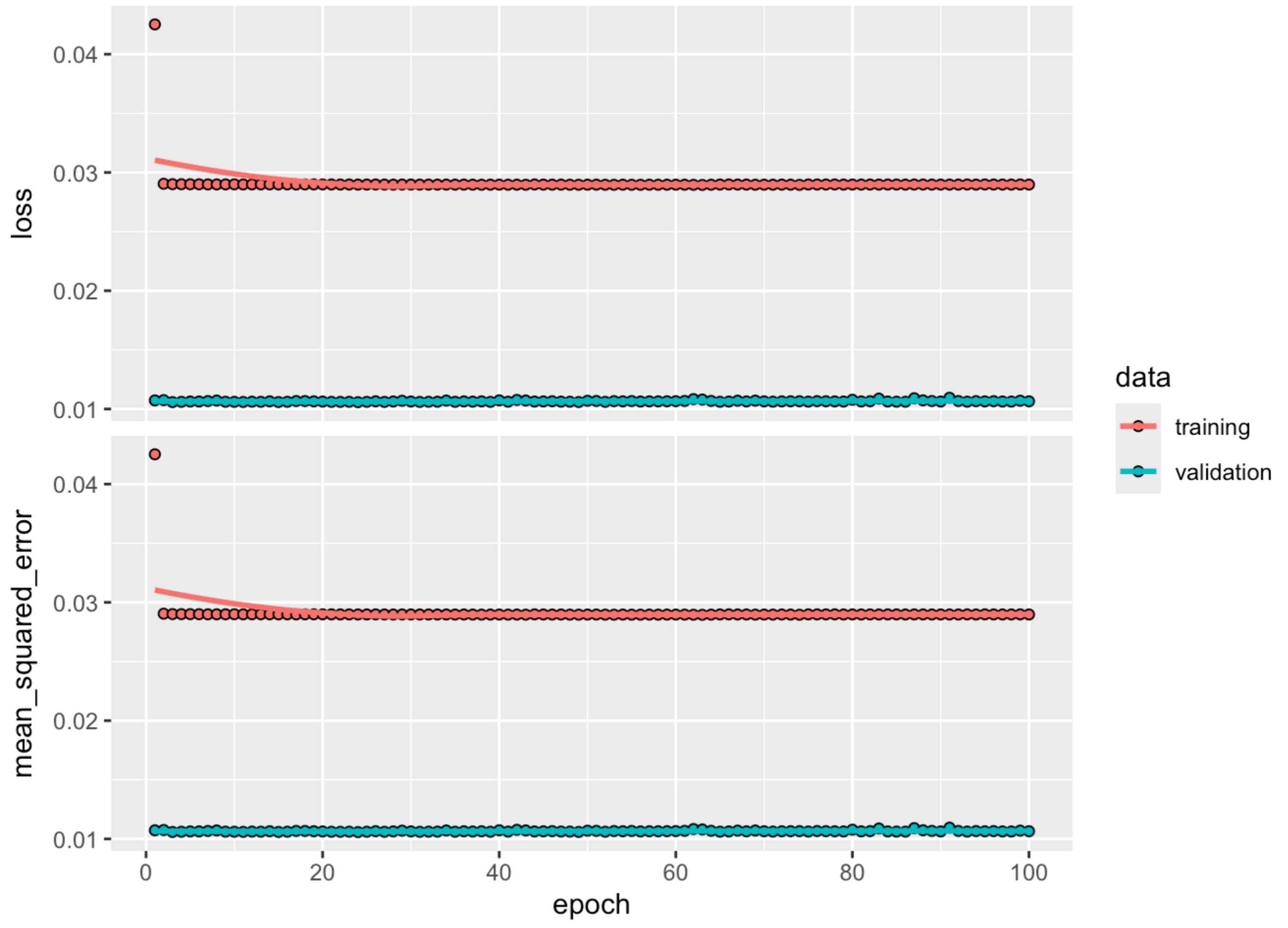
```
# SPCA transformed matrices for training, validation and test
Z_train_spca <- X_train_spca %*% spca_result$loadings
Z_val_spca   <- X_val_spca %*% spca_result$loadings
Z_test_spca  <- X_test_spca %*% spca_result$loadings
```

```
# Define a grid of hyperparameters
units1_list <- c(16, 32, 64)
units2_list <- c(8, 16, 32)
learning_rates <- c(0.001, 0.0005)
batch_sizes <- c(64, 128)

results <- data.frame()

for (units1 in units1_list) {
  for (units2 in units2_list) {
    for (lr in learning_rates) {
      for (bs in batch_sizes) {

        # Build model
        model <- keras_model_sequential() %>%
          layer_dense(units = units1, activation = "relu", input_shape = ncol(Z_train_sp
ca)) %>%
          layer_dense(units = units2, activation = "relu") %>%
          layer_dense(units = 1)

        # Compile
        model %>% compile(
          loss = "mse",
          optimizer = optimizer_adam(learning_rate = lr),
          metrics = list("mean_squared_error")
        )

        # Train
        history <- model %>% fit(
          x = Z_train_spca,
          y = y_train_nn_spca,
          epochs = 100,
          batch_size = bs,
          validation_data = list(Z_val_spca, y_val_nn_spca),
          verbose = 0
        )

        # Get final validation MSE
        val_mse <- tail(history$metrics$val_mean_squared_error, 1)

        # Record result
        results <- rbind(results, data.frame(
          units1 = units1,
          units2 = units2,
          learning_rate = lr,
          batch_size = bs,
          val_mse = val_mse
        ))
      }
    }
  }
}
```

```
# Find the best hyperparameters
best_result <- results[which.min(results$val_mse), ]
print(best_result)
```

```
##    units1 units2 learning_rate batch_size    val_mse
## 13     32      8         0.001         64 0.01060999
```

```
# Train NN on SPCA components
model_spca <- keras_model_sequential() %>%
  layer_dense(units = best_result$units1,
              activation = "relu", input_shape = ncol(Z_train_spca)) %>%
  layer_dense(units = best_result$units2, activation = "relu") %>%
  layer_dense(units = 1)  # output layer for regression


# Compile model
model_spca %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(learning_rate = best_result$learning_rate),
  metrics = list("mean_squared_error")
)

history_spca <- model_spca %>% fit(
  x = Z_train_spca,
  y = y_train_nn_spca,
  epochs = 100,
  batch_size = best_result$batch_size,
  validation_data = list(Z_val_spca, y_val_nn_spca),
  verbose = 0  # quiet training
)

plot(history_spca)
```



```
# Predict on test set
pred_spca_train <- model_spca %>% predict(Z_train_spca)
```

```
## 6192/6192 - 1s - 918ms/epoch - 148us/step
```

```
pred_spca_test <- model_spca %>% predict(Z_test_spca)
```

```
## 853/853 - 0s - 131ms/epoch - 154us/step
```

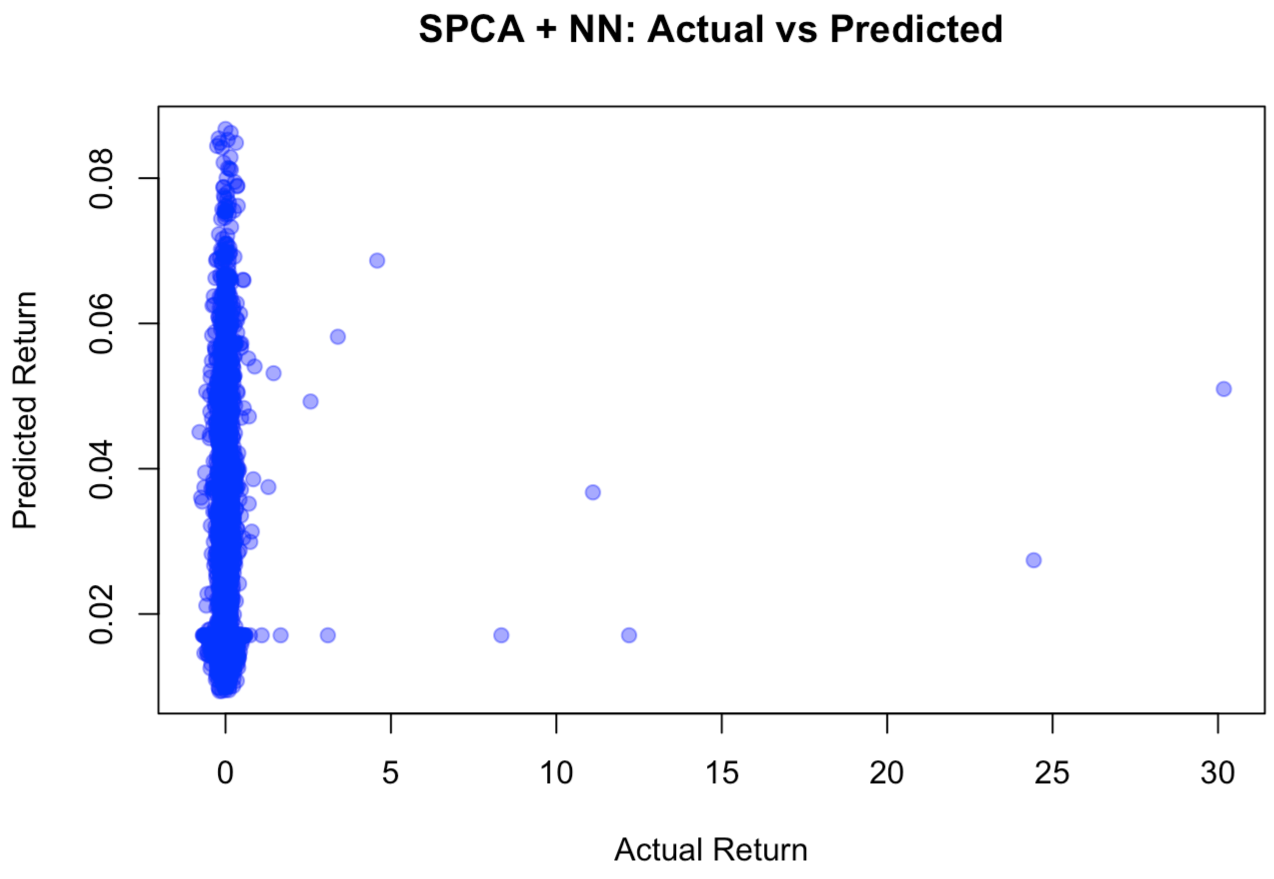
```
# Calculate RMSE
rmse <- function(actual, predicted) {
  sqrt(mean((actual - predicted)^2))
}

cat("SPCA + NN Train RMSE:", rmse(y_train_nn_spca, pred_spca_train), "\n")
```

```
## SPCA + NN Train RMSE: 0.1702056
```

```
cat("SPCA + NN Test RMSE:", rmse(y_test_nn_spca, pred_spca_test), "\n")
```

```
## SPCA + NN Test RMSE: 0.2802199
```

```
plot(y_test_nn_spca, pred_spca_test,
     xlab = "Actual Return",
     ylab = "Predicted Return",
     main = "SPCA + NN: Actual vs Predicted",
     #xlim = c(-0.05, 0.05),   # Zoom on X-axis: Actual returns between -10% and +10%
     # ylim = c(-0.1, 0.1),
     pch = 19, col = rgb(0, 0, 1, 0.4))
```



```r
# Add predictions back to test set (already done if continuing)
testing_set_lasso <- testing_set %>%
  mutate(pred_return = as.numeric(pred_nn_lasso_test),
         model = "LASSO_NN")

testing_set_spca <- testing_set %>%
  mutate(pred_return = as.numeric(pred_spca_test),
         model = "SPCA_NN")

# Combine both models
portfolio_data <- bind_rows(testing_set_lasso, testing_set_spca)

# Rank stocks within each date and model
portfolio_data <- portfolio_data %>%
  group_by(model, date) %>%
  mutate(rank = percent_rank(pred_return)) %>%
  ungroup()

# Filter top 20% (long-only portfolio)
top_20_data <- portfolio_data %>%
  filter(rank >= 0.8)

# Calculate average monthly return
long_only_returns <- top_20_data %>%
  group_by(model, date) %>%
  summarise(top20_return = mean(target_return), .groups = "drop")

# Performance summary
long_only_summary <- long_only_returns %>%
  group_by(model) %>%
  summarise(
    avg_monthly_return = mean(top20_return, na.rm = TRUE),
    holding_period_return = prod(1+top20_return)-1,
    sd_monthly_return = sd(top20_return, na.rm = TRUE),
    sharpe_ratio = avg_monthly_return / sd_monthly_return,
    .groups = "drop"
  )

print(long_only_summary)
```

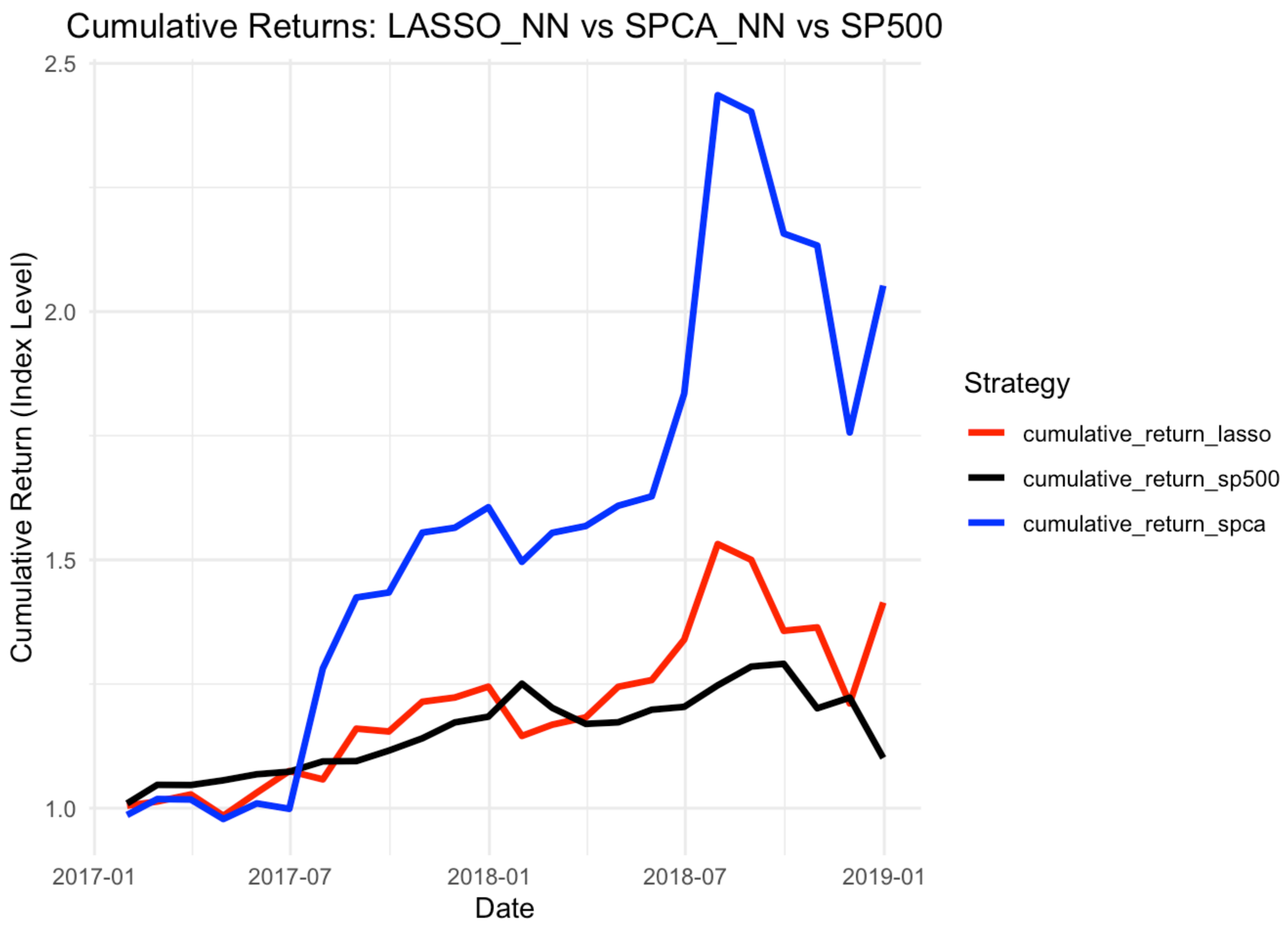
```
## # A tibble: 2 × 5
##   model  avg_monthly_return holding_period_return sd_monthly_return sharpe_ratio
##   <chr>               <dbl>                 <dbl>             <dbl>        <dbl>
## 1 LASSO…             0.0165                 0.414            0.0650        0.254
## 2 SPCA_…             0.0357                 1.05             0.109         0.326
```

```
# Calculate
sp500_avg_return <- mean(sp500_monthly_returns$sp500_return, na.rm = TRUE)
sp500_sd_return <- sd(sp500_monthly_returns$sp500_return, na.rm = TRUE)
sp500_holding_period_return <- prod(1+sp500_monthly_returns$sp500_return) -1
sp500_sharpe <- sp500_avg_return / sp500_sd_return

# Show result
tibble(
  model = "SP500",
  avg_monthly_return = sp500_avg_return,
  holding_period_return = sp500_holding_period_return,
  sd_monthly_return = sp500_sd_return,
  sharpe_ratio = sp500_sharpe
)
```

```
## # A tibble: 1 × 5
##   model avg_monthly_return holding_period_return sd_monthly_return sharpe_ratio
##   <chr>              <dbl>                 <dbl>             <dbl>        <dbl>
## 1 SP500            0.00458                 0.101            0.0339        0.135
```

```
# cumulative returns for three strategies
long_only_returns_lasso <- long_only_returns[long_only_returns[,'model'] == 'LASSO_NN',]
long_only_returns_spca <- long_only_returns[long_only_returns[,'model'] == 'SPCA_NN',]

# Add cumulative returns
monthly_portfolio_return_lasso <- long_only_returns_lasso %>%
  mutate(cumulative_return_lasso = cumprod(1 + top20_return))

monthly_portfolio_return_spca <- long_only_returns_spca %>%
  mutate(cumulative_return_spca = cumprod(1 + top20_return))

sp500_monthly_returns <- sp500_monthly_returns %>%
  mutate(cumulative_return_sp500 = cumprod(1 + sp500_return))

# mutate sp500 date column to make sure it aligns with the date from lasso and spca
# Replace sp500 date with spca portfolio date
sp500_monthly_returns <- sp500_monthly_returns %>%
  mutate(date = long_only_returns_lasso$date)

compare_cumulative_returns <- monthly_portfolio_return_lasso %>%
  select(date, cumulative_return_lasso) %>%
  left_join(monthly_portfolio_return_spca %>% select(date, cumulative_return_spca), by =
"date") %>%
  left_join(sp500_monthly_returns %>% select(date, cumulative_return_sp500), by = "dat
e")


compare_cumulative_returns_long <- compare_cumulative_returns %>%
  pivot_longer(
    cols = starts_with("cumulative_return"),
    names_to = "model",
    values_to = "cumulative_return"
  )

ggplot(compare_cumulative_returns_long, aes(x = date, y = cumulative_return, color = mod
el)) +
  geom_line(size = 1.2) +
  labs(title = "Cumulative Returns: LASSO_NN vs SPCA_NN vs SP500",
       x = "Date",
       y = "Cumulative Return (Index Level)",
       color = "Strategy") +
  theme_minimal() +
  scale_color_manual(values = c("cumulative_return_lasso" = "red",
                                "cumulative_return_spca" = "blue",
                                "cumulative_return_sp500" = "black")) +
  theme(plot.title = element_text(hjust = 0.5))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```


Cumulative Returns: LASSO_NN vs SPCA_NN vs SP500
Cumulative Return (Index Level)
Date
Strategy
cumulative_return_lasso
cumulative_return_sp500
cumulative_return_spca