ML-based US stock return prediction and asset allocation

Tianze Shou
Carnegie Mellon University
Pittsburgh, PA 15213
tshou@andrew.cmu.edu

Wenhan Li
Carnegie Mellon University
Pittsburgh, PA 15213
wenhanli@andrew.cmu.edu

Lucy Hu
Carnegie Mellon University
Pittsburgh, PA 15213
yaoyunh@andrew.cmu.edu

Mia Zhang
Carnegie Mellon University
Pittsburgh, PA 15213
sizhez@andrew.cmu.edu

Abstract

Predicting return rate in the US stock market has always been a heavily investigated and financially rewarding realm. A variety of Machine Learning methods have also been frequently employed for this exact purpose in the recent years. In this study, we use historical stock data from the CRSP dataset since the year 1980 to train a family of stock return predictors, including elastic net regressor, random forest regressor, neural network regressors, and logistic classifier. Inspired by previous works, we propose novel neural network architecture and method to transform the classic stock return regression task to classification task [1]. We evaluate the model performance with simulated trading, where each model takes long position on the top 10% decile of the stocks ranked by predicted return and shorts the bottom 10%. The overall portfolio return is calculated on a value-weighted basis. We find that the logistic classifier model dominates the long-position-only portfolios, while the elastic net model attains the highest wealth within long-short portfolios. All models' performance exceeds the S&P500 market index [2].

1 Introduction

Within the last 40 years, more and more complex and rapid information channel has been established between the US stock market and market participants, allowing the market itself to be more transparent and efficient. Various market participants have also been practicing quantitative trading strategies to handle this information explosion. Such methods can effectively help traders understand the potential return rate and corresponding risks of their portfolios. In modern days, more weights have been shifted to the machine learning components of those quantitative strategies: hundreds of thousands of signals (hand-crafted features) are collected and used every day as predictors of the market; and a variety of models, ranging from a simple linear model to more complex neural networks, are employed in attempt to make accurate predictions of future stock prices. Therefore, fitting historical stock data to make predictions of the future is of great economic value, and we believe that a context like the US stock market is well-suited for building and training deep neural networks.

[1] Source code for all experiment is available in this public repository

[2] We thank Prof. Lars-Alexander Kuehn and Mingjun Sun for directing our work

36-490 Class Project (2023).

2 Related Work

2.1 Data source

Our data of analysis is the Center for Research in Security Prices (CRSP) dataset as described by Bali et al. [2016]. The data collects cross-sectional stocks in the US from NYSE, AMEX, and NASDAQ together with several hundred hand-crafted features (*signals*) since December 31, 1925. Each entry in the data table consists of one publicly traded stock, observation date, observed features, and cross-sectional excess return rate of the stock. Our main target of prediction, excess return rate, is defined as:

$$\alpha = R_s - R_b \tag{1}$$

where R_s is the return rate of the stock over a given time period, and R_b is the benchmark return rate of a riskless investment (e.g., government bond). The excess return (α) essential describes how much more return one can achieve in comparison with investing in other fixed-income securities.

2.2 ML in finance

Gu et al. [2018] gives an exhaustive overview of machine learning methodology being applied to stock return prediction and evaluates their performances within the extremely noisy environment. The most conventional methods involve building simple linear models with or without regularization terms. Those models include linear regression models with Lasso, Ridge regularization, or a combination of both (i.e., elastic net) in order to prevent overfitting by reducing the models' degree of freedom. Others have attempted to use tree-based models (e.g., decision trees) and ensemble learning methods (e.g., random forest and adaptive boosting).

However, none of the above models give powerful predictions of the market. The R^2 metric of the above model all ranges from 0.1% to 0.3% in the CRSP dataset. This is partly due to 1) the fact that the market is highly efficient and difficult to predict in nature and 2) the simple model does not have a sufficient degree of freedom to utilize a large number of signals. Bagnara [2022] has mentioned using dimension reduction methods, PCA in particular, to solve the second problem, but the tested performance does not show a significant effect.

2.3 Asset allocation strategies

We also would like to trade stocks after predicting their return rates. Similar to the momentum strategy proposed by Jegadeesh and Titman [1993], the modern common approach is to take a long position (buy) of stocks with top 10% predicted excess return and take a short position of stocks with bottom 10% predicted return, which is discussed by Poh et al. [2020]. In short, we would like to purchase the highest-growth stocks and sell the worst-performing ones. Also note that the return rate can be negative, and shorting stocks with negative returns will yield positive gains.

We will evaluate the performance of our constructed portfolio via value-weighted methods proposed by Fama and French [1992]. We calculate the overall return of the portfolio using the following:

$$r_p = \frac{\sum_i r_i Q_i}{\sum_i Q_i} - \frac{\sum_j r_j Q_j}{\sum_j Q_j} \tag{2}$$

where r_p is the excess return rate of the entire portfolio; r_i and Q_i represent the return and total market share of stock i that we are taking a long position; and j represents the stocks we are taking a short position. The weighted portfolio return will then be compared with the S&P500 index as a benchmark.

3 Methodology

3.1 Training structure

As discussed in section 2.1, we use the CRSP dataset to train our return prediction models. Since stock data is less accurate and exhaustive in the early 20th century, we will only use the stock observations after the year 1980. Each data entry will consist of arbitrary stock A's observed cross-sectional features (X) at time t and A's excess return rate (y) with regard to the previous observation at time $t-1$. X is a 21-dimensional vector representing 21 hand-crafted financial features after rank transformation following the same methodology as Gu et al. [2018].

Due to the time-series nature of the stock data, we train each model class under the same rolling training structure. In essence, for each year of data we trade on, or in other words, conduct tests on, we will train a unique model with the previous years' data. More specifically, we define a *training cycle* of be the interval of time-series data that we use to train-validate-test a model. Suppose T is the start of a training session, we use the interval $[T, T+10)$ for training, interval $[T+10, T+15)$ for validation, and interval $[T+15, T+16)$ for testing. Note that the unit of time here is *one year*. After the completion of the current training cycle, we assign $T \leftarrow T+1$ and repeat. All models will follow the same training cycle as visualized in figure 1 to avoid bias.

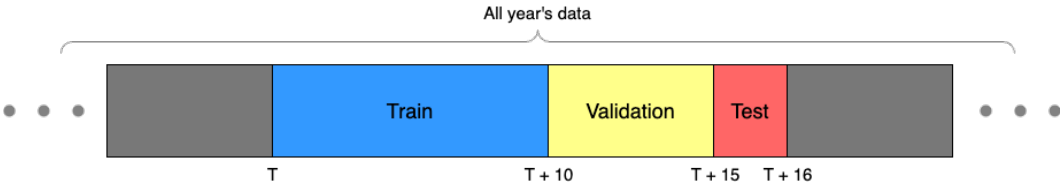

Figure 1: Train-validation-test structure

Different from the training and validation process where we pool all data in the interval disregarding their sequence in time, we separate the one-year test data into 12 slices (one for each month) and run inference on them separately. Thus, we are able to obtain monthly predictions and also simulate monthly trading.

3.2 Elastic net

As mentioned in section 2.2, we underscored the widespread adoption of linear regression models as the standard for forecasting excess return rates. This is because in the context of stock return prediction, understanding the rationale behind a prediction can be as valuable as the prediction itself. Stakeholders often seek clarity on the factors influencing excess return rate, as this not only builds trust in the model but also enables more informed investment decisions. Such insights are often more readily available from linear models than from complex neural networks. Therefore, we intentionally choose to implement a rolling elastic net regression model as the foundational benchmark.

Elastic Net regression combines the strengths of Ridge regression's parameter shrinkage and Lasso regression's feature selection to limit the model's degree of freedom. The objective function for Elastic Net regression can be expressed as:

$$RSS + \lambda * [(1-\alpha) * ||\beta||_2 + \alpha * ||\beta||_1] \tag{3}$$

where RSS is the residual sum of squares, λ is the regularization parameter, β is the coefficient vector, and α is the mixing parameter between L1 and L2 norms. To optimize our model, a grid search is executed on the validation data across both hyperparameters, leading to the identification of the optimal parameters. More specifically, we construct a two-dimensional grid search for the optimal λ and α values, where candidate λ values range in $\{0, 10^{-4}, 10^{-3}, ..., 10^{7}\}$, and candidate α values range in $\{0, 0.1, 0.2, ..., 0.9, 1\}$. The grid search mechanism will then train an elastic net model on the training data using each combination from the candidates. The optimal model will be selected from the candidate models with the highest R^2 score on the validation data. The model, equipped with these best-fit hyperparameters, is then used to predict the outcomes on the test dataset on a monthly basis.

3.3 Random forest regressor

Continuing our exploration of models for excess return predictions, we turn to the random forest regressor model, which offers a more complex, non-linear approach compared to the model discussed in 3.2. Random forest is an ensemble learning method that constructs multiple decision trees for training and outputs the mean prediction of individual trees. Our model uses the random forest regressor from the scikit-learn library and controls a few parameters: number of trees in the forest (`n_estimators = 100`), max depth of a tree (`max_depth = 1`), max number of features at a split (`max_features = 7`), and max fraction of dataset to be drawn for training (`max_samples = 0.5`). With the above parameters specified, we use a one-dimensional grid search for the optimal minimum number of samples for a split, where the range is $\{5, 8, 11, 14, 16\}$. Similar to 3.2, the model is trained on a rolling basis and evaluated using R^2 score. Post-validation, the model with the highest R^2 score is selected for final evaluation on the test dataset.

3.4 Neural networks

We acknowledge that the simple linear model and tree-based ensemble model described in sections 3.2 and 3.3 may lack predictive power due to their linear nature and lack of deep, latent representation of input data. Since stock features/signals may have a non-linear relationship with their returns, inserting deep, hierarchical, and latent representation structures may enable our model to make more accurate predictions. Inspired by Gu et al. [2018], we first implement a feedforward neural network with three hidden layers (NN3) with input and output dimensions being 21 and 1, and each hidden dimension being 32, 16, and 8 as shown in figure 2.

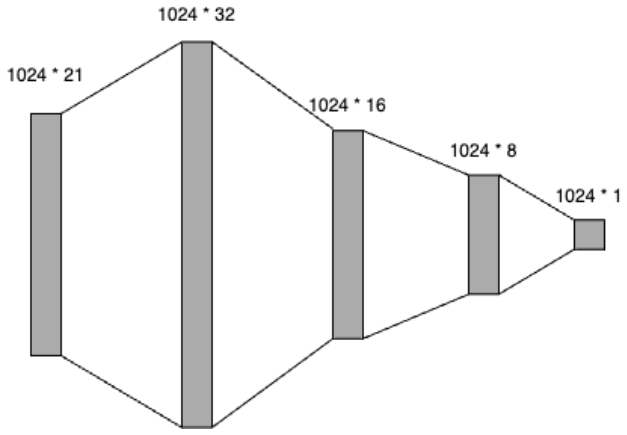

Figure 2: Model architecture of NN3

We follow the same train-validation-test structure as other models. We optimize the neural network weights for 50 epochs (this is shown to be sufficient with experiments as validation performance only improves marginally after around 20 epochs) and select the optimal model from the epoch with the highest validation R^2. For parameter optimization, we use the mean squared error loss function and stochastic gradient descent algorithm with a learning rate of 0.001 and momentum of 0.9 to optimize the network weights.

Gu et al. [2018] has shown that neural networks with five hidden layers are not necessarily better performing than three-layer networks in the context of stock trading. We suspect that this is due to 1) model overfitting causing the neural network to pick up noise in the training data, 2) neural network degradation as more layers are added. To address these two issues, we apply the dropout technique proposed by Srivastava et al. [2014] and the residual link technique proposed by He et al. [2015]. Following these ideas, we propose a neural network with five hidden layers (each of dimension 32) with dropout layers (dropout probability is 0.15) after each hidden layer and the input layer and residual link enabling the model to skip each hidden layer. This results in the model architecture shown in figure 3. The training and optimization method for NN5 is the same as NN3.

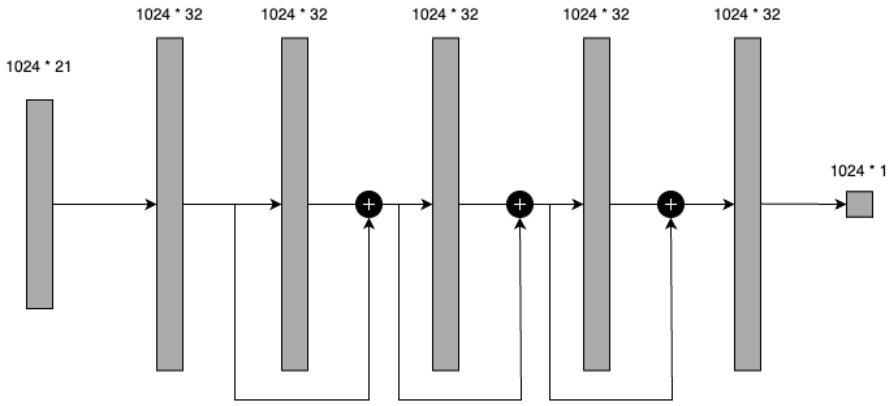

Figure 3: Model architecture of NN5

3.5 Logistic classifier

Besides regression models, we also give a touch on the classification model. The model we select is logistic regression model. Logistic regression model had already been repeated used in the finance corporate. Before training the model, we transfer the original continuous response variable into 10 categories by assigning them to percentiles based on their distribution within each month. Specifically, returns in the top 10% are categorized as 10, the next 10% as 9, and so on, down to the lowest 10%. The rationale behind this is that we only need the stocks with first and the last ten percentile excessive returns in the asset allocation. So instead of the logistic regression model that would yield binary response variables, the logistic regression model we use is a multinomial logistic regression that is capable of predicting more than 2 classes of response variables.

Multinomial logistic regression extends the concept of logistic regression to accommodate response variables with more than two categories. This statistical technique models the probability of each category as a function of the independent variables. Besides, in order to reduce the risk of overfitting, we also add a L1 regularization penalty to the model. The objective function for multinomial logistic regression is typically a log-likelihood function:

$$L = -\sum_{i=1}^{N}\sum_{k=1}^{K}[y_{ik}\log(p_{ik})] + \lambda\sum_{j=1}^{M}|\beta_j| \tag{4}$$

Here, the first term represents the log-likelihood loss for multinomial distribution, where N is the number of observations, K is the number of categories, y_{ik} is a one-hot encoded label of whether observation i belongs to category k, and p_{ik} is the predicted probability of observation i belonging to category k. The second term is the L1 penalty, where λ is the regularization strength, and β_j are the coefficients of the model.

With the predictions from our logistic model, we could easily get the stocks with first and the last ten percentile of excessive returns for our asset allocation. However, we also acknowledge that the classification model usually does not have the attribution of easy interpretability like linear regression or elastic nets. Unlike these simpler models where the relationship between variables can be directly quantified, classification models often function as "black boxes". Besides, in classification, the discrete categories do not always convey a clear, intuitive sense of gradation or relationship between the predictor variables and the outcome. However, we still experiment with this model to explore how effectively classification models can handle the complex and nuanced nature of our dataset.

4 Results

After completing the model training, we proceeded to obtain the excess returns for the next month for each individual stock. These returns were then integrated into our asset allocation strategy, as detailed in section 2.3. Our analysis commenced with an initial investment of $1 in 1995, allowing

us to track how this initial capital evolved over time when applying these strategies across various models. We also plotted the baseline S&P500 market index as a benchmark.

Figure 4 represents the progression of portfolio growth over time when employing various models, specifically focusing on a long-only strategy (i.e., purchasing only the top 10% of stocks). Furthermore, Figure 5 illustrates the portfolio's growth when utilizing different models and adopting both long and short positions in the market. In this case, we bought the top 10% of stocks while simultaneously selling short the bottom 10% of stocks.

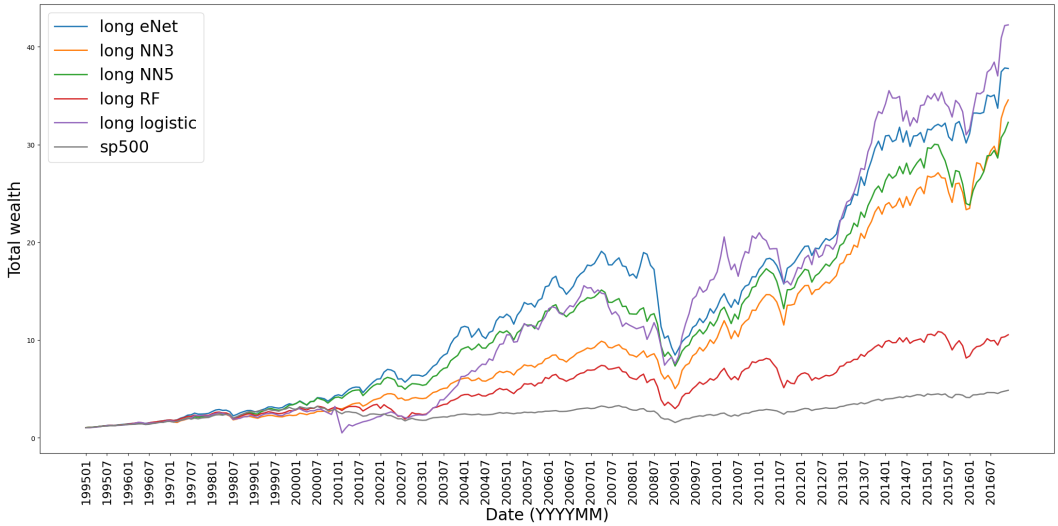

Figure 4: Portfolio growth for long-only portfolios

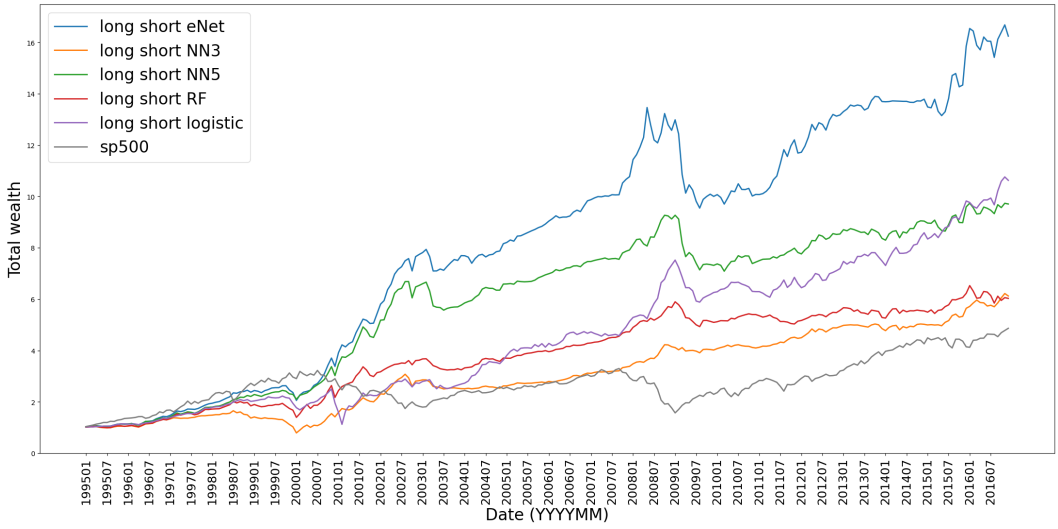

Figure 5: Portfolio growth for long-short portfolios

As shown in Figure 4, the long-only strategy utilizing logistic regression stands out as the top performer, followed by the elastic net model and the neural network with three layers. Remarkably, the initial $1 investment grew to approximately $43 by the year 2016, while the baseline S&P 500 portfolio only reached around $4.5 during the same period. This substantial difference underscores the fact that the logistic regression model has outperformed the baseline by approximately 9 times.

Now, turning our attention to Figure 5, it is evident that the elastic net model performs best, followed by logistic regression and the neural network with five layers. However, the overall performance of employing only long positions remains superior to taking both long and short positions. Even with

the best-performing model in this category, the portfolio only grew to approximately $17 by 2016, which is about 4 times better than the baseline.

It is noteworthy that in both Figure 4 and Figure 5, all models exhibited poor performance during 2008, which aligns with the overall economic conditions. This performance decline coincides with the significant financial crisis that occurred during that period. However, it's interesting to observe that the drop in portfolio value in Figure 5 (long-short) was less pronounced than the drop in Figure 4 (long-only). This difference suggests that employing both long and short positions in the portfolio is more stable and fluctuates less.

This outcome is reasonable, considering that during the 2008 financial crisis, many stocks performed very poorly. Therefore, selling these underperforming stocks would likely have resulted in a more favorable portfolio outcome than holding onto them. This observation underscores the potential benefits of adopting a more flexible approach that includes both long and short positions, particularly during turbulent economic periods.

5 Discussion and Analysis

5.1 Sharpe Ratio as validation metric

Upon evaluating our work, we have identified a few limitations that could be improved on in future research. Primarily, our use of the R^2 score as the primary metric for hyperparameter tuning has certain drawbacks. In the context of financial modeling, relying solely on statistical metrics may not necessarily lead to the optimal selection of hyperparameters for our models. Therefore, we want to introduce a financial metric during hyperparameter tuning: the Sharpe Ratio proposed by Sharpe [1964].

The Sharpe Ratio plays a crucial role in assessing the performance of an investment, such as a security or portfolio, relative to a risk-free asset. It is defined as:

$$S_a = \frac{E[R_a - R_b]}{\sigma_a} \tag{5}$$

By trying to maximize the Sharpe Ratio during hyperparameter tuning, we anticipate that this approach will yield models better suited to the nuances and demands of the financial setting, ultimately leading to more effective and robust results.

5.2 Stock sequence modeling

We also realize that our model only utilizes the immediate cross-sectional stock feature observation to make a return prediction, disregarding previous observations of the same stock. In order to remedy this, we would like to include the possibility of fitting a sequence model, such as RNN (Jordan [1986]), on our data.

Suppose an arbitrary stock A holds n historical cross-sectional feature observations $X^1, X^2, ..., X^n$ where the superscript denotes the time step. We also define an RNN with sequence length L, where only the final block outputs the prediction of the return rate. We can then use the RNN to conduct autoregressive-like training where we use $X^i, ..., X^{i+l}$ to make a prediction for y^{i+l+1}, $X^{i+1}, ..., X^{i+l+1}$ for y^{i+l+2}, etc. One can also simply replace the vanilla RNN architecture with LSTM blocks (Hochreiter and Schmidhuber [1997]) in order to be more robust against vanishing/exploding gradient problems.

We choose to temporarily hold on to this idea because 1) sequence models lack a theoretical foundation in the financial realm, 2) we have not come up with a mechanism that can integrate autoregressive sequence models with our existing training framework. Therefore, we hope to implement and experiment with this idea in the future.

References

Matteo Bagnara. Asset pricing and machine learning: A critical review. *Journal of Economic Surveys*, 2022. doi: 10.1111/joes.12532.

Turan G. Bali, Robert F. Engle, and Scott Murray. *The CRSP Sample and Market Factor*. John Wiley & Sons, Inc., 2016.

Eugene F. Fama and Kenneth R. French. The cross-section of expected stock returns. *The Journal of Finance*, 47(2):427–465, 1992.

Shihao Gu, Bryan T. Kelly, and Dacheng Xiu. Empirical asset pricing via machine learning. *SSRN Electronic Journal*, 2018. doi: 10.2139/ssrn.3281018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Narasimhan Jegadeesh and Sheridan Titman. Momentum strategies. *The Journal of Finance*, 48(3): 979–1007, 1993.

M I Jordan. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986. 5 1986. URL `https://www.osti.gov/biblio/6910294`.

Daniel Poh, Bryan Lim, Stefan Zohren, and Stephen Roberts. Building cross-sectional systematic strategies by learning to rank. *SSRN Electronic Journal*, 2020. doi: 10.2139/ssrn.3751012.

William F. Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk*. *The Journal of Finance*, 19(3):425–442, 1964. doi: https://doi.org/10.1111/j.1540-6261.1964.tb02865.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1964.tb02865.x`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1): 1929–1958, jan 2014. ISSN 1532-4435.