UNIVERSITY OF CALIFORNIA,
IRVINE

Applications of Machine Learning and Reinforcement Learning in Investment and Trading

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Management

by

Chenzhe Tian

Dissertation Committee:
Associate Professor Zheng Sun, Chair
Professor Lu Zheng
Assistant Professor Jinfei Sheng

2020

DEDICATION

To

my wife Xueqi Tian 田雪琪,

who brings me light in the dark and makes me a better man,

in recognition of her unconditional support and love.

To

our forthcoming son Mancang Tian 田满仓

We wish you love, peace, happiness, courage, and wisdom.

吉士思秋 实感物化 日与月与 荏苒代谢 逝者如斯 曾无日夜 嗟尔庶士 胡宁自舍

研精耽道 安有幽深 安心恬荡 栖志浮云 如彼南亩 力耒既勤 藨蓘致功 必有丰殷

川广自源 成人在始 累微以著 乃物之理 若金受砺 若泥在钧 进德修业 辉光日新

——【西晋】 张华

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express the deepest appreciation to my committee chair, Professor Zheng Sun, who gave me the freedom and guidance to explore my academic area of interest. Without her guidance and persistent help this dissertation would not have been possible.

I would like to thank my committee members, Professor Lu Zheng and Professor Jinfei Sheng, for their kindness to share thoughts and comments on my research ideas.

In addition, a thank you to Professor Ye Mao of University of Illinois at Urbana-Champaign, who introduced me to research in Finance, and whose enthusiasm for the "market micro-structures" had lasting effect.

I thank my PhD fellows who have been journeyed with me in recent years. Those exciting moments we discuss research and life would stay in my mind forever.

I thank Paul Merage School of Business for the support of funding of my PhD study.

CURRICULUM VITAE

Chenzhe Tian

2011	B.S. in Chemical Engineering, Tsinghua University
2013	M.S in Financial Engineering, University of Illinois at Urbana-Champaign
2020	Ph.D. in Finance, University of California, Irvine

FIELD OF STUDY

Applications of Artificial Intelligence in Empirical Asset Pricing and Financial Intermediaries

ABSTRACT OF THE DISSERTATION

Applications of Machine Learning and Reinforcement Learning in Investment and Trading

By

Chenzhe Tian

Doctor of Philosophy in Management

University of California, Irvine, 2020

Professor Zheng Sun, Chair

Machine learning is increasingly gaining applications in Finance industry. In this dissertation, I use machine learning methods to predict mutual fund and hedge fund performances and address the issue whether mutual fund and hedge fund managers add value. Overall, machine learning methods tend to outperform OLS in terms of return prediction. From a machine learning point of view, mutual fund managers don't add value while hedge funds do deliver risk-adjusted performance. Also, such outperformance of top hedge funds is persistent with three-year horizon. Furthermore, such outperformance provided by machine learning methods are not driven by fund characteristics. A regression of machine learning outperformance on macroeconomic variables show that machine learning models tend to perform better when the economy is in recession, when the market is bearish, and when the market and economic policy uncertainty are increased.

Recently, reinforcement learning (RL) is being explored by practitioners in trading. In particular, algorithmic trading provides an ideal setting for RL. In this dissertation, I trained an agent to place aggressive stock market orders with deep Q-network with multi-step temporal difference. The back-test results show that, the aggressive order agent outperforms an agent with linear execution schedule by an average of 0.12 to 0.69 basis points on simulated orders in Asia Pacific stock markets. This shows that RL-based methods are capable of recognizing and utilizing market information and are promising to outperform traditional execution algorithms.

In summary, machine learning and reinforcement learning are promising to provide better performances in portfolio investment and trading.

Introduction

With the breakout of the big data era and the recent development of machine learning, the finance industry is experiencing fundamental changes. For example, when customers apply for credit, machine learning algorithms in the back end is helping lenders evaluate their credit worthiness according to applicants' own historical records and data of others with similar characteristics. Also, there are billions of credit card transactions every day, identifying a few fraud transactions among billions is like finding needles in haystacks. With the help of machine learning, this task could be done within seconds. There are countless examples of machine learning applications that are making our life more efficient.

Although machine learning has been successfully applied in consumer banking and fraud detection, it's application in the area of investments, especially to predict mutual fund and hedge fund returns, seems very scarce. In fact, in mutual fund and hedge fund literature, researchers mainly use linear models. However, due to the complexity of economic activities, using linear models to characterize the fundamental associations between returns and predictive variables is far from enough. One of the biggest advantages of machine learning is the ability to review large volumes of data to identify patterns and trends, and such patterns and trends are often non-linear.

Furthermore, in mutual fund literature, the question whether mutual fund managers add value is a long-existing debate. With linear model such as ordinary least squares as the tool, researchers have reached opposite conclusions with different data. One of the possible reasons for such contradiction is the lack of tools that are capable of capturing subtle and non-linear relationships in the data. With the help of machine learning, we may be able to get closer to the fundamental relationships and add more evidences to the debate from the point of view of machine learning.

Another research topic that machine learning could be helpful is in hedge fund performance. Although researchers have generally accepted that hedge fund managers are able to deliver positive risk-adjusted returns, whether such performances are persistent is not clear. Machine learning, with the ability to uncover non-linear relationships, could be helpful to unveil the mystery.

Another fast-evolving technology that is changing the way we live is reinforcement learning. Its applications are already everywhere in our life. Our new cars may be equipped with self-driving functionality that can go to any place without human intervention. Such achievement is done by training a computer program on large amount of existing data, rather than explicitly coding rules in the program. When you are calling customer service, sometimes you can't tell whether it's a human or a machine on the other end. Such naturality is based on voice recognition and natural language processing. And perhaps astonishing, the best chess player in the world is not human anymore, but a computer program with self-improvement ability that has seen and learnt millions of chess games, which is not possible to be achieved by human players. Behind the scene of these moments in life is tremendous amount of data and the successful development and application of reinforcement learning. However, when it comes to trading of financial products, especially in those highly computerized environments, there is very few reinforcement learning applications. Hence, the cross discipline of reinforcement learning and finance is a whole new area of research that needs to be explored.

In the first two chapters of this dissertation, I will explore the possibility to use machine learning methods to find a systematic way to predict mutual fund and hedge fund returns. For mutual funds, the general conclusion is that machine learning methods are promising to provide higher raw returns than linear method for a zero-investment portfolio to take long positions in top performers and short positions in bottom ones. However, such performance is not statistically significant after risk adjustment. Hence, mutual fund managers do not add value from the angle of machine learning.

On the other hand, for hedge fund data, using the same machine learning methods, I do find higher raw returns than linear method, as well as risk-adjusted performances for the zero-investment strategy. Such outperformance of machine learning methods is persistent with three-year horizon and is robust under the presence of hedge fund characteristics. Furthermore, in a regression analysis of the outperformance with macroeconomic variables, I find that machine learning methods tend to outperform when the economy is in recession, when the market is bearish, and when the uncertainty increases.

In the third chapter, I explore the possibility to use reinforcement learning in the setting of stock trading. Specifically, I train a computerized agent with deep Q-network and multi-step temporal difference that can execute aggressive orders under various market conditions. With a linear execution schedule as the benchmark, the agent can systematically beat the benchmark by 0.12 to 0.69 basis points in 11 Asia Pacific stock markets. Furthermore, the outperformance is robust under high noise-to-signal ratios.

To my knowledge, my findings make the following contributions to finance literature:

1. this is the first paper to explore the possibility to use machine learning methods to predict mutual fund and hedge fund returns;

2. it adds a new piece of evidence that mutual fund managers do not add value from a machine learning point of view;

3. it demonstrates that, when applied to hedge fund data, machine learning methods are promising to generate higher risk-adjusted performances and persistence than linear methods;

4. this is the first paper to apply reinforcement learning in the area of aggressive stock trading. And it demonstrates that the performance of reinforcement learning-based agent is promising to beat a linear benchmark, even under the presence of high noise-to-signal ratios, which is valuable to industry practitioners.

Overall, this dissertation demonstrates the possibility to apply cutting edge technologies in machine learning and reinforcement learning in the area of investments and trading.

Chapter 1

On the Skills of Mutual Fund Managers via Machine Learning Methods

1.1 Background

Mutual fund is an area with abundant research in finance. However, some fundamental questions, such as whether mutual fund managers can generate positive excess returns, or whether their performance is persistent, remain drastically debated. On the one hand, some researchers believe that mutual fund managers do not add value. One of the earliest papers with this argument dates back to Jensen (1968), which finds that mutual funds on average are not able to outperform a buy-the-market-and-hold strategy and that individual funds cannot perform better than what is expected from mere random chance. Sharpe (1966) finds evidence that supports the efficient markets by denying the ability of fund managers to beat a risk-adjusted market portfolio. Elton, Gruber, Das, and Hlavka (1993) find that mutual funds do not earn returns that justify their information acquisition costs, which implies the net-of-fee alpha provided by mutual funds is negative. Carhart (1997) suggests that one-year momentum can explain the "hot hand" phenomenon described in Hendricks, Panel, and Zeckhauser (1993).

On the other hand, some researchers argue that mutual fund managers are capable of generating risk-adjusted returns. Grinblatt and Titman (1992) find positive persistence in mutual fund performance. Hendricks, Panel, and Zeckhauser (1993) find a "hot hand" effect within top performers. Wermers (2000) finds that funds' stock picking ability enables them to cover their costs, thereby supporting the value of active mutual fund management. Ibbotson and Patel (2002) also find performance persistence in mutual funds. Also, Berk and Green (2004) provide a rational model of active portfolio management and argues that lack of persistence does not imply lack of skill. They argue that fund flows pursue past superior performance even though performance is not persistent. With diminishing returns to scale, in equilibrium superior funds do not show persistence. Pástor, Stambaugh, and Taylor (2015) confirm decreasing returns to scale at both industry level and fund level and the active management industry has become more skilled over time.

There are two potential channels for the above-mentioned papers to reach opposite conclusions: lack of comprehensive data and possible incapability of research methodology. Since the origination of the debate, mutual fund data has become more comprehensive. Researchers has found various variables that has predictive power of future fund performances. For example, Chevalier and Ellison (1997) find a non-linear flow-performance relationship and that mutual fund managers alter the riskiness of their portfolios to take advantage of such relationship. Kacperzyk, Sialm, and Zheng (2005) create an industry concentration index and finds that more concentrated funds perform better. Kacperzyk, Sialm, and Zheng (2008) find that return gap, which is defined as the difference between the reported fund return and the return on a portfolio that invests in the previously disclosed fund holdings, predicts fund performance. Cremers and Petajisto (2019) introduce Active Share as a measure of activeness of mutual funds and finds that funds with the highest Active Share significantly outperform their benchmarks. Amihud and Goyenko (2013) find that a low level of coefficient of determination, which is the R^2 obtained from a regression of fund returns on a multifactor benchmark model, significantly predicts better performance.

Furthermore, most of the above-mentioned papers use ordinary least squares (OLS) method to evaluate the relationship between performance and predictive variables. However, OLS can only pick up linear relationships. If the relationship is non-linear, it could be ignored by OLS or even mis-specified. Recently machine learning has drawn much attention from both industry and academia and has found many successful real-world applications such as predicting credibility of borrowers and algorithmic trading. Many machine learning models are better than OLS in prediction in that machine learning models are better at reviewing large volumes of data and identifying patterns and trends. Therefore, using machine learning models to predict mutual fund performance with more comprehensive data is a natural research topic to pursue.

In this chapter, I will first evaluate the potential effectiveness of seven machine learning models, including least absolute shrinkage and selection operator, ridge regression, principal component regression, partial least squares, random forest, gradient boosting random trees, and neural networks, in a simulation with non-linear predictive variables, and select the models with the highest predictabilities. Next, the selected models will be applied to a comprehensive mutual fund data with 94 independent variables to predict fund returns. Then a portfolio sorting approach will be used to evaluate the absolute performance and risk-adjusted performance of the machine learning models. Finally, the debate whether mutual fund managers have skills will be revisited.

The rest of this chapter is organized as follows. Section 1.2 will give a brief introduction of machine learning models used in this chapter. Section 1.3 presents the fund-level variables used as independent variables, the methodologies to back-test model performance, and performance evaluation measures. Section 1.4 presents data and summary statistics. Section 1.5 shows empirical evidence of tests. And section 1.6 concludes.

1.2 Machine Learning Methods

In this section, I will introduce seven machine learning methods that are used in this chapter: least absolute shrinkage and selection operator (LASSO), ridge regression, principal component regression (PCR), partial least square (PLS), decision trees, random forests (RF), gradient boosting random tree (GBRT), and neural networks (NN). Here I only focus on the basic ideas of these models, so that readers without machine learning and programming backgrounds can easily follow.

1.2.1 The Basics of Machine Learning

Machine learning, often seen as a subset of artificial intelligence, includes a set of algorithms or statistical models. It is often used to perform prediction tasks. Machine learning models are “trained”, i.e., fitted or calibrated, on sample data in order to make predictions without being explicitly programmed to perform the task. According to the nature of the task, machine learning falls into two categories: supervised learning and unsupervised learning.

In supervised learning, models are trained on sample data, which includes both predictor as inputs and desired results as outputs. In the language of machine learning, the inputs are also called features, and the outputs are called labels or targets. Supervised learning can be further categorized into two main types according to the nature of targets: regression and classification. A regression model tries to predict targets with continuous values. For example, given historical firm characteristics such as size, book-to-market ratio, or momentum, predict future stock price. Or given build year, location, and size, predict house prices. In these two examples, both stock price and house price are continuous variables, therefore, these two tasks are regression problems. A classification model, on the other hand, tries to predict a categorical target. For example, given health records such as a computed tomography image, predict whether a patient has cancer or not. Or given credit history and financial ratios of a borrower, predict whether it will default in the future.

In unsupervised learning, data does not have a label, i.e., there is no specific or desired output to predict. Rather, we often use unsupervised learning algorithms to perform grouping or clustering tasks. In practice, supervised learning makes the majority of machine learning tasks, which is also the focus of this chapter.

A major difference between machine learning and traditional linear models such as OLS is that the structure of machine learning models often is much more complicated. As a result, machine learning models are much richer in terms of parameters, which makes them more flexible than traditional linear models. The optimal values of model parameters are inferred and gradually adjusted during training process. Another characteristic that makes machine learning models different from OLS is that they almost always have hyper-parameters. Hyper-parameters are a set of parameters that control the model complexity, regularization, training process, and stopping criteria, etc. Unlike model parameters, hyper-parameters, cannot be inferred during training. Instead, they can only be set before training starts. And the optimal values of hyper-parameters are obtained through trial-and-error with multiple times of training.

Although machine learning models are very flexible, there are downsides as well. The biggest issue is over-fitting, which is the problem that a model fits too closely or exactly to a particular set of data but fails to fit additional data. To control over-fitting, sample data is often split into three exhaustive and mutually exclusive sets, i.e., training set, validation set, and test set. Training set is the set that machine learning models are trained on. Given the large number of parameters, machine learning models can be easily fit to the training set with arbitrarily good in-sample performance, which leads to over-fitting. This is where regularization comes into play. Regularization is the process of imposing additional requirements on the model parameters to discourage the model from becoming overly complex.

To control the extent of regularization, the validation set are used to tune hyper-parameters in order to obtain a good out-of-sample performance at the cost of reducing in-sample performance. However, it's also possible that machine learning models are over-fitted to the validation set. In order to diagnose whether this happens, we also need the test set to evaluate the performance of machine learning models on samples that haven't been used in training, nor hyper-parameter tuning, which are truly out-of-sample. Ideally, a well-trained model should have similar performance on all three sets. Good training set performance with bad validation and test set performance suggest that the model is over-fitted to training set. And good training and validation set performance with inferior test set performance indicate the model is over-fitted to the validation set.

Finally, to train a machine learning model, we also need an objective or loss function that measures the difference between ground truth (i.e., true target values) and model predictions. For a continuous target, mean-squared-error or mean-absolute-error are often used as the loss function. For a categorical target, cross entropy is often used. From a mathematical point of view, training the model is just to minimize the loss by adjusting model parameters, subject to any imposed regularization criteria.

Next, I will introduce machine learning models used in this chapter one by one.

1.2.2 Least Absolute Shrinkage and Selection Operator (LASSO)

LASSO, introduced in Tibshirani (1996), is a shrinkage method in the sense that it shrinks some of the regression coefficients completely to zero. Simply speaking, LASSO imposes a penalty term to OLS loss as the sum of the absolute value of coefficients. Its Lagrangian form is

$$\hat{\beta}^{LASSO} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \left| \beta_j \right| \right\},$$

where λ is the regularization parameter, which controls the extent of penalty that is imposed to non-zero coefficients. The first term on the right-hand-side is the OLS loss, and the second term is LASSO regularization penalty. Note that this absolute-value penalty has very nice mathematical property such that given an appropriate value of λ, it shrinks some of the regression coefficients completely to zero. Therefore, LASSO does a kind of variable selection. Obviously, OLS is just a special case of LASSO with the regularization parameter $\lambda = 0$. A subtle but important technical point is that LASSO is scale variant. Therefore, when using LASSO, the features should be standardized first to have zero mean and unit variance, such that all features are on the same scale. This is important because standardization ensures that the coefficients are penalized fairly on the same scale.

The LASSO estimator does not have an explicit analytical solution. To solve LASSO numerically, we can use Least Angle Regression introduced in Efron, Hastie, Johnstone, and Tibshirani (2004).

1.2.3 Ridge Regression

Tikhonov regularization or ridge regression, introduced in Tikhonov (1963), is another shrinkage method. Ridge shrinks the coefficients of regression by imposing a penalty on their size. Mathematically, the solution of ridge regression minimize a (squared size of coefficients) penalized residual sum of squares:

$$\hat{\beta}^{ridge} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right) + \lambda \sum_{j=1}^{p} \beta_j^2 \right\},$$

where $\lambda \geq 0$ is the parameter that controls the amount of regularization: larger values of λ result in greater amount of shrinkage. As a result, higher λ shrinks β more towards zero. Note that unlike LASSO, which causes some of the coefficients to be exactly zero, ridge is a kind of "soft" shrinking, which does not cause the coefficient to be zero exactly. Therefore, in ridge regression, every input variable contributes to the loss function. Finally, similar to LASSO, the features should be standardized first to ensure that ridge coefficients are penalized fairly on the same scale.

Ridge regression estimator has analytical solution, which is

$$\hat{\beta}^{ridge} = (X^T X + \lambda I)^{-1} X^T Y.$$

1.2.4 Principal Components Regression (PCR)

Unlike LASSO or ridge regression that use the standardized features as input, PCR uses the principal components of the input features as the independent variables in OLS. The principal components Z_m is computed as $Z_m = Xv_m$, where X is the input features, and v_m is the eigenvector of the variance-covariance matrix of X. Since Z_m are all orthogonal to each other, the regression with M principal components is just a sum of product between univariate regression coefficients $\hat{\theta}$ and the principal components Z:

$$\hat{y}_M^{PCR} = \hat{\theta}_0 + \sum_{m=1}^{M} \hat{\theta}_m Z_m,$$

where by construction $\hat{\theta}_0 \equiv \bar{y}$.

A subtle technical point to note is that PCR is scale variant. Thus, before PCA decomposition, the input feature X also need to be standardized, since PCA projects the original data onto directions that maximize the variance. Without standardization, the features with largest variance will mechanically compose a principal component, thus suppressing the effect of other components and reducing the efficiency of PCR.

1.2.5 Partial Least Squares Regression (PLS)

Similar to PCR, PLS also uses a linear combination of inputs for regression. However, unlike PCR that constructs inputs (components) only with input features X, PLS also uses target y. To perform PLS, y is univariately regressed on each feature X_i to obtain a univariate coefficient $\beta_{1,i}$, which works as a partial sensitivity. Then the first regressor Z_1 is computed as $Z_1 = \frac{\sum_i \beta_{1,i} X_i}{\sum_i \beta_{1,i}}$, which is the weighted average of X_i's. And then we regress y on Z_1. Next X is orthogonalized with respect to Z_1, and the above process is repeated until the desired number of regressors are obtained. By construction, if we repeat this process as many times as the number of features, we will restore the solution of OLS. Similar to PCA, PLS is also scale variant, so we need to standardize the input features before conducting PLS regression.

1.2.6 Decision Trees

Although decision tree is not used as a separate estimator in this chapter, it is the building block of random forest and gradient boosting random trees. A decision tree is a tree-like decision flow, in which non-leaf nodes represent the conditions/features to split on, and leaf nodes represent a portion of targets. For example, in classification trees the leaf nodes represent one of the target categories, whereas in regression trees the leaf nodes represent a specific target value. More specifically, a decision tree "learns" the data in an iterative fashion. In each iteration, the tree selects the feature (grows a branch) that splits current set of observations optimally according to some measure such as impurity. Within each branch, the tree again selects the feature that splits the observations in the current branch optimally. Essentially, a decision tree repetitively divides the feature space (potentially a hyper-space if there are more than 3 features) into smaller sub-spaces according to one or multiple threshold values of features, and it keeps doing so until all the subspaces are pure, i.e., the labels of each observation in the same subspace are the same. However, in practice it's often very hard to obtain a pure partition. Therefore, the decision tree is stopped according to some stopping criteria. For example, the tree stops growing if, a) the total depth of the tree reaches a predefined value, or b) there is no branch with number of observations more than a prespecified value, or c) the number of leaves reaches a predetermined number, etc. For more details about decision trees, please refer to Breiman, Friedman, Olshen, and Stone (1984).

Obviously, the learning process of decision tree makes it extremely prone to over-fitting to training data and perform poorly on new data. Therefore, in practice a single decision tree is hardly used. Instead, random forest and gradient boosting methods, which are derivatives of decision trees, are developed to overcome such disadvantage of decision trees.

1.2.7 Random Forest

As its name suggests, random forest consists of multiple decision trees. Simply speaking, each decision tree in a random forest is trained simultaneously on a bootstrapped sample with replacement from the training data, such that trees with different bootstrapped sample grow differently. At this point, each tree is still prone to over-fitting to its bootstrapped sample. However, when it comes to predicting new/unseen samples, each tree in the random forest is used to make a prediction, and the final prediction of the random forest is made by taking a majority vote of the trees in case of a classification problem, or by taking the average of the predictions of individual trees in case of a regression problem. In this way, over-fitting is largely mitigated. Therefore, by increasing the number of trees in the random forest, out-of-sample performance is often improved. However, the marginal improvement is decreasing with the number of trees growing, and a larger random forest requires more computational resources. But the trees in the random forest can be estimated in parallelization.

1.2.8 Gradient Boosting

Unlike random forest that trains each individual tree in a parallel fashion and takes a majority vote or an average, the idea of gradient boosting is to sequentially connect multiple weak learners to form a single strong learner in an iterative fashion. The logic behind the scene is that subsequent learners learn from the mistakes of their predecessors. In the first iteration, a simple and shallow tree is trained on the training data. This tree almost surely fails to achieve a good performance since its shallow structure makes it underfit to the training data, hence the term *weak learner*. In the second iteration, another simple decision tree is trained on the residuals of the prediction of the first decision tree, trying to correct the errors of its predecessor. Then the prediction of the first two trees is summed up to form a slightly better prediction, and a set of new residuals between the ground truth and this new prediction is also computed. In the third iteration, a new simple tree is trained on this newly computed set of residuals. And similarly, a new set of predictions and residuals are computed as a result. The model keeps growing one tree after another until the prespecified number of trees are reached or some other stopping criteria is fulfilled. Gradient boosting is fairly robust to over-fitting, so a large number of trees usually results in better performance. However, similar to random forest, gradient boosting with more trees requires more computational power. And the computation of the individual trees cannot be done in parallelization.

1.2.9 Neural Networks

Neural network, or artificial neural network, is a computation system that is partly inspired by biological neural networks. The building block of a neural network is neuron. A neural network is consisted of multiple layers, and each layer is consisted of multiple neurons. There are connections between neurons in consecutive layers, but no connection exists between neurons in the same layer. A neuron is essentially a non-linear transformation of its input from a mathematical point of view. Inside a neuron, there are two stages of operations. The first stage is a linear transformation of the inputs, i.e., $y = WX + b$, where W is called a weight matrix and b is called. Both the weight matrix and the bias are learnable parameters of the network. In the second stage, a non-linear activation function $\sigma(\cdot)$ is applied to the linearly transformed inputs, i.e., $\mathrm{a} = \sigma(y) = \sigma(WX + b)$. Several choices of the activation function are available, including sigmoid function, hyperbolic-tangent function, rectified linear unit or ReLU, etc. The role of these activation functions is to provide non-linearities for the network. Then the nonlinearly transformed output a is passed to the next connected neuron as its input.

The simplest neural network consists of three layers: one input layer, one output layer, and one hidden layer in the middle. The neurons in the input layer get their values from the features of data and pass such values directly to the hidden layer, hence the name *input layer*. By construction, the number of neurons in the input layer is the same as the number of features. The hidden layer gets its values from the input layer, after nonlinearly transforming the inputs, it passes those values to the output layer. For a regression problem or a binary classification problem, there is generally only one neuron in the output layer. For a non-binary classification problem, the number of neurons in the output layer is the number of categories in the target. The output layer then uses its inputs to make a prediction.

The number of parameters grows extremely fast with number of hidden layers and number of neurons in each layer. Also, a deep neural network (neural networks with many hidden layers) often requires a huge amount of data to train. To give readers a sense of magnitude, one of the most successful deep neural network architectures for image recognition, called GoogLeNet, has 22 layers to classify objects in images into 1000 categories. This network has about 100 million parameters and is trained with 1.2 million images.

To train a neural network, we impose a loss function $L \equiv L(y_{pred}, y_{true}; W, b)$ that measures the difference between the prediction of the neural network and the ground truth. Therefore, the ultimate goal of the network is to get the value of the output layer as close to the target as possible, by changing the value of network parameters W and b. As a result, the loss function of the neural network is differentiable with respect to W and b. A method called gradient descent is used to minimize the loss function. More specifically, the loss function L is minimized iteratively with the so-called forward-propagation step and backward-propagation step. In forward propagation, a set of predictions are computed with the current parameter values W_i and b_i. Then in the back-propagation step, the loss function is evaluated and differentiated w.r.t. W_i and b_i to compute the partial derivative $\frac{\partial L}{\partial W_i}$ and $\frac{\partial L}{\partial b_i}$. Afterwards, W and b are updated in the following way: $W_{i+1} = W_i - \alpha \cdot \frac{\partial L}{\partial W_i}$, and $b_{i+1} = b_i - \alpha \cdot \frac{\partial L}{\partial b_i}$, in which α is the learning rate. The updated W and b are again used in forward-propagation and a set of new partial derivatives is also computed in the following backpropagation. Thus forward-propagation and backward-propagation alternate to minimize the loss function.

1.3 Methodology

1.3.1 Fund-Level Performance Predictors

The existing mutual fund literature has documented a variety of fund characteristics that have a statistically significant effect on mutual fund's performance. In the following subsections I will present the basic ideas and formal definitions of the variables used in this chapter as fund-level performance predictors.

1.3.1.1 Fund Flow

The relationship between flow and performance is a heavily addressed topic in mutual fund literature. Sirri and Tufano (1998) find a non-linear relationship between fund flow and past performance: mutual fund investors chase returns by flocking to funds with highest previous returns, though failing to punish poorly-performing fund managers by withdrawing funds. Berk and Green (2004) argue that there is decreasing returns to scale, and performance-chasing behavior of investors drives mutual fund performance towards zero in equilibrium. Therefore, past performance and flows has indirect effects on future performances. In this chapter, we follow the definition of fund flow in Sirri and Tufano (1998):

$$FLOW_{i,t} = \frac{TNA_{i,t} - TNA_{i,t-1} * (1 + R_{i,t})}{TNA_{i,t-1}},$$

where $TNA_{i,t}$ is fund i's total net assets at time t, and $R_{i,t}$ is the fund's return over the period ending at time t.

1.3.1.2 Industry Concentration Index

Kacperczyk, Sialm, and Zheng (2005) argue that mutual fund managers may have informational advantages on industries they are familiar with, and accordingly deviate their holdings from a well-diversified portfolio and concentrate on such industries. The authors show that mutual funds with higher industry concentration index (ICI) perform better than others, even controlling for risk. ICI is defined as below:

$$ICI_t = \sum_{j=1}^{10} (\omega_{j,t} - \bar{\omega}_{j,t})^2,$$

where $\omega_{j,t}$ is the fund's weight on industry j at time t, and $\bar{\omega}_{j,t}$ is the weight on industry j at time t for the market portfolio. Therefore, ICI measures how much a mutual fund's portfolio deviates from the market portfolio.

1.3.1.3 R^2 Selectivity

Amihud and Goyenko (2013) propose that a mutual fund's performance can be predicted by its R^2, which is the coefficient of determination from a regression of its returns on a multifactor benchmark model, such as Carhart four-factor model. Higher R^2 indicates a fund tracks the benchmark closely. On the contrary, lower R^2 means the fund tracks the benchmark less closely. Thus, R^2 selectivity is defined as $1 - R^2 \equiv \frac{SSE}{SST}$, where SSE is the residual sum of squares, and SST is the total sum of squares. Therefore, the R^2 selectivity measures how much of a fund's variance of returns can be attributed to idiosyncratic risk.

1.3.1.4 Return Gap

Kacperczyk, Sialm, and Zheng (2008) argue that the actions of mutual fund managers are not fully observed by investors. In order to estimate the impact of such unobserved actions, they calculate the return gap, i.e., the difference between the net return on recently disclosed portfolio holdings and fund's reported return. The authors find that funds with large past return gaps tend to outperform, even after adjusting for risks and investment styles. Therefore, this project adopts their methodology to construct return gap as below.

Given a fund's recent portfolio holdings, its hypothetical "return on holdings" is

$$RH_t^f = \sum_{i=1}^{n} \tilde{\omega}_{i,t-1}^f R_{i,t},$$

where RH_t^f is the return of holdings at time t for fund f, $\tilde{\omega}_{i,t-1}^f$ is the weight of asset i from the most recent portfolio disclosure for fund f, and $R_{i,t}$ is asset i's return at time t. Note that funds may have changed their holdings during the period from $t-1$ to t, such action is unobservable, however, hence this is a hypothetical return. To compute the net return of holdings, the expense ratio of fund f is subtracted from RH_t^f. As a result, the return gap is defined as

$$RG_t^f = RF_t^f - (RH_t^f - EXP_t^f),$$

where RF_t^f is the fund's reported return, and EXP_t^f is the expense ratio at time t for fund f.

1.3.1.5 Active Share

Cremers and Petajisto (2009) introduce a measure to gauge the activeness of mutual fund managers, by comparing the weight of holdings in mutual fund portfolio with the weight of corresponding assets in benchmark indexes. The authors find that mutual funds with high active share measure significantly outperform their benchmark indexes, while those with low active share measure underperform. The active share is defined as the following:

$$Active\ Share = \frac{1}{2}\sum_{i=1}^{N} \left|\omega_{fund,i} - \omega_{index,i}\right|,$$

where $\omega_{fund,i}$ is the weight of asset i in the fund's portfolio, and $\omega_{index,i}$ is the weight of asset i in the benchmark index.

1.3.1.6 Other Fund-Level Performance Predictors

Hendricks, Patel, and Zeckhauser (1993) show that lagged one-year return has predictive power for future one-year returns. Also, Carhart (1997) shows that expense ratios and turnover are negatively related future performance, and past alpha from four-factor model also has some predictive power for future risk-adjusted performance. Furthermore, Chen, Hong, Huang, and Kubik (2004) show that actively managed funds present decreasing returns to scale for various performance benchmarks, due to illiquidity of small stocks. Finally, I use fund age as a performance predictor to control for incubation bias (Evans (2010)), though later in Section 1.4 I require a mutual fund have at least three years of return history to enter my sample.

1.3.2 Back-Testing Methodology

1.3.2.1 Cross Validation

Traditionally, for OLS models, to evaluate the performance we often split a sample into two mutually exclusive sub-samples, fit OLS on one sub-sample and test the OLS on another sample. However, as discussed previously, machine learning models often are rich in parameters and the model structures are very flexible, which make them prune to overfitting. To overcome such issue, various types of regularization are used. These regularization methods are implemented through model hyper-parameters, which directly controls model structures. Therefore, for machine learning methods, a sample is split into three sub-samples instead of two, i.e., training sample, validation sample, and test sample.

The training sample is used to fit the model, validation sample is used to tune model hyper-parameters, and test sample is used to evaluate model performance. In this chapter, I use K-fold cross validation to tune model hyper-parameters and an out-of-sample data set to evaluate model performance. The procedures are as following.

A sample is first split into two subsets: training set and test set. The training set would have the majority of the observations. Next a set of hyper-parameters is pre-set to determine model structure and the extent of regularization. Then, the training set is further evenly split into K folds. For each fold i, the model will be trained on all folds except fold i, and the performance will be evaluated on fold i. Hence, we will have one performance value for each fold, and the overall performance associated with the current set of hyper-parameters is just the average of these values. We can repeat these procedures with a different set of hyper-parameters until we reach a good performance. After the set of optimal hyper-parameters is obtained, the performance of the model associated with such set of hyper-parameters will be evaluated on the test set to verify the efficacy of the trained model.

1.3.2.2 Back-Testing

My data is panel data with monthly frequency ranging from 1983 to 2017. To evaluate the overall performance, I use expanding rolling samples, and recalibrate model monthly. Starting from January 1998, in each month, I use all of the previous data for cross-validation with three folds. And the model performance is evaluated for the current month. For example, to evaluate the model performance in October 2000, all models would be cross validated with three folds on data from January 1983 to September 2000.

Besides the above time-series back-testing procedure, I also adopt the back-testing methodology introduced in Mamaysky, Spiegel, and Zhang (2007). The basic idea is to only predict the performance on those mutual funds for which the model has good performance on in the past. The procedures are as follows:

1) Estimate a model on data up to $t-2$.
2) Use the obtained model in step 1) to calculate predicted returns at $t-1$ for each fund. For each fund, if the sign of the predicted return is the same as realized return, then it's added to the active pool.
3) Refit the model only on funds in the active pool on data up to $t-1$, and evaluate model performance at t.

1.3.3 Performance Measures

In this chapter, I use three measures to gauge model performance: root mean squared error, ordinary R^2, R^2 defined in Gu, Kelly, and Xiu (2018) (GKX R^2). The root mean squared error is defined as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \widehat{y_i})^2}{N}}$$

And the normal R^2 is defined as follows:

$$1 - \frac{\sum_{i=1}^{N}(y_i - \widehat{y_i})^2}{\sum_{i=1}^{N}(y_i - \overline{y_i})^2}$$

The last performance measure is GKX R^2, which is defined and used by Gu, Kelly, and Xiu (2018). Unlike normal R^2 which assume a model that always predicts average y as the benchmark, the GKX R^2 assumes a model that always predicts 0 as the benchmark.

$$1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} {y_i}^2}$$

In the next section, I will present the sample construction process and summary statistics.

1.4 Data and Summary Statistics

For my empirical analysis, I create the mutual fund data set by merging four databases: Center for Research in Security Prices (CRSP) U.S. stock database, Standard & Poor's Capital IQ COMPUSTAT, CRSP Survivorship-Bias-Free Mutual Fund Database, and Thomson Reuters CDA/Spectrum Mutual Fund Holdings (S12). The CRSP U.S. stock database contains end-of-day and end-of-month information, including high, low, close, volume, returns, etc., for stocks listed on NYSE, NYSE MKT, NASDAQ, and Arca exchanges. The COMPUSTAT database provides more than 500 company-level fundamentals on income statements, balance sheets, and statement of cash flows, which is mainly collected from SEC filings. The CRSP mutual fund database includes fund-class-level information, including monthly returns, total net assets (TNA), expense ratios, front/rear loads, turnover ratio, and other fund characteristics. The CDA/Spectrum database provides stock holdings data starting in the first quarter of 1980 for mutual funds and investment companies, which are also collected from SEC filings. Prior to 1985, SEC requires mutual funds to report their portfolio holdings on a quarterly basis. After 1985, mutual funds are required to report semi-annually. However, a slight majority of mutual funds also voluntarily report their quarterly holdings. Around 2000, mutual funds are required to report portfolio holdings on a quarterly basis again. Next, I briefly describe the procedures to create the data set.

My mutual fund data set includes three types of performance predictors: fund-level characteristics, stock-level characteristics, and macro-economic variables. I focus on open-end U.S. domestic equity mutual funds, for which data are most reliable and complete, and eliminate balanced, bond, index, international, and sector funds. I follow Wermers (2000) to merge CRSP mutual fund database and CDA/Spectrum Mutual Fund Holdings database. To avoid incubation bias (Evans (2010)), I require mutual funds to have at least 36 months return history. Following Elton, Gruber, and Blake (1996), I require mutual funds to have total net assets of at least $15 million to avoid survivorship bias caused by small funds. Following Kacperczyk, Sialm, and Zheng (2005), I require all mutual funds to have at least 11 stock holdings in order to have a reasonable industry concentration index (ICI) measure.

The original holdings data is at quarterly frequency (a minority of the observations are at semi-annual frequency). Since mutual fund holdings are reasonably stable and my research objective is focused on predicting monthly mutual fund returns, I populate holdings data from quarterly frequency to monthly frequency by assuming that portfolio holdings remain the same (shares held remain the same and relative weight changes with stock prices) since the most recent holdings report, i.e., mutual funds only rebalance their portfolio at the end of their fiscal quarter. For each fund-month, holdings are linked to CRSP stock database and COMPUSTAT to collect and compute fund-level and stock-level characteristics.

In this chapter, I collected and computed 14 fund-level characteristics that have been shown to have a significant effect on fund performance in the existing literature, including expense ratio, turnover ratio, Carhart alpha based on the previous 36 month performance (Carhart (1997)), age (Evans (2010)), fund size (Chen, Hong, Huang, and Kubik (2004) and Berk and Green (2004)), prior one-year return (Hendricks, Patel, and Zeckhauser (1993)), previous one-month net flow, previous one-quarter net flow, previous one-year net flow (Berk and Green (2004) and Sirri and Tufano (1998)), industry concentration index (ICI) (Kacperczyk, Sialm, and Zheng (2005)), R-square selectivity (Amihud and Goyenko (2013)), return gap, previous one-year average return gap (Kacperczyk, Sialm, and Zheng (2008)), and active share (Cremers and Petajisto (2009)).

SEC requires mutual funds to report portfolio holdings within 60 days of the end of their fiscal quarter, to avoid look-ahead bias, all holdings-related fund-level characteristics, for example return gap, ICI, and active share, are created with two-months lag.

Besides fund-fund level characteristics, following Green, Hand, and Zhang (2013), I also use 71 stock-level characteristics that have been shown to have predictive powers for stock returns[1]. One thing to note is that in their original paper, the authors use 94 stock-level characteristics. However, in my data set I found some of the variables are extremely highly correlated. To reduce unnecessary data redundancy and to improve model interpretability, especially for the linear-based models, these 94 predictors are preprocessed through a stepwise procedure call VIF test: eliminate one predictor with the highest variable inflation factor (VIF) at a time until the VIF of all predictors are below a threshold of 5. After this procedure, 71 stock-level predictors remain, and all correlation coefficients are below 0.8. These stock-level predictors are merged with holdings data for each fund-month, and then collapsed to fund level by averaging across all holdings with the

[1] Thanks to Professor Jeremiah Green for kindly sharing the SAS code to construct these predictors.

value of holdings as weight. Similar to fund-level characteristics, all 71 predictors are lagged by 2 months.

In addition, following Welch and Goyal (2008), I also construct eight macroeconomic predictors, including dividend-to-price ratio, earning-to-price ratio, book-to-market ratio, net equity expansion, treasury-bill rate, term spread, default spread, and stock variance at monthly frequency[2].

My final mutual fund sample spans from 1983 to 2017. Table 1.1 reports summary statistics of the fund-level characteristics. There are 3596 distinct funds and 464702 fund month observations in total. The number of distinct funds in each month ranges from 180 (April 1983) to 2000 (September 2008), with an average of 1147. From 1983 to 2017, an average fund earns a monthly raw return of 0.89% with a slightly negative risk-adjusted return of -0.01% (Carhart (1997) four-factor). For mutual fund total net assets, the average is about $1461 millions, with a median of only $332 millions and a standard deviation of about $3500 millions, suggesting the fund size is extremely positively skewed: a minority number of funds manage a majority value of assets.

Table 1.2 reports the correlation coefficients of monthly returns and its predictors. The correlations between lagged flow and previous one-year return, and the correlations between lagged flow and risk-adjusted past performance are all moderately positive (for example, correlation of one-year flow and past 36-month Carhart alpha is 0.4), confirming the performance-chasing behavior of mutual fund investors. Also, there is a high correlation of 0.84 between ICI and R^2 selectivity.

[2] The data used to construct these macro variables is available at Professor Amit Goyal's website.

Table 1.1 Summary Statistics of Mutual Fund Data

Variable	Mean	Median	Standard Deviation	Minimum	Maximum
Total number of funds	3596				
Average number of funds per month	1147				
Total number of observations	464702				
Monthly raw return	0.89%	1.26%	5.03%	-36.70%	40.00%
Monthly Carhart alpha	-0.01%	-0.01%	1.89%	-21.54%	35.82%
Previous 1-year return	11.40%	12.85%	20.72%	-73.64%	251.02%
Carhart alpha of past 36 months	0.04%	0.02%	0.37%	-3.34%	2.52%
Total net assets (in millions)	1461.39	331.90	3488.48	15.00	41764.70
Expense ratio	1.14%	1.13%	0.45%	0.00%	3.67%
Age	15.07	11.59	11.67	3.00	62.33
Past 1-month flow	-0.02%	-0.50%	5.87%	-71.78%	244.70%
Past 1-quarter flow	1.72%	-0.86%	20.22%	-83.84%	640.34%
Past 1-year flow	11.16%	-4.98%	78.08%	-128.98%	2958.03%
Turnover ratio	79.04%	58.00%	81.39%	0.90%	1153.00%
Industry concentration index	8.31%	3.78%	14.25%	0.02%	89.85%
R^2 selectivity	10.24%	7.23%	10.50%	0.06%	99.15%
Return gap with 2 lags	-0.03%	-0.02%	0.98%	-17.25%	17.81%
12-month average return gap with 2 lags	-0.03%	-0.03%	0.32%	-3.02%	2.58%
Active share with 2 lags	83.00%	88.96%	19.51%	5.42%	100.00%

Table 1.2 Cross-Sectional Correlation of Fund-Level Characteristics

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)
(1) Monthly Raw Ret	1.00															
(2) Carhart Alpha	0.50	1.00														
(3) Previous 1-yr Ret	-0.09	-0.08	1.00													
(4) Past 36m Carhart alpha	-0.23	-0.11	0.03	1.00												
(5) TNA	-0.03	-0.04	0.00	-0.03	1.00											
(6) Exp Ratio	-0.12	-0.13	-0.13	-0.11	-0.66	1.00										
(7) Age	-0.04	-0.07	-0.08	-0.25	0.44	-0.41	1.00									
(8) Flow 1M lag1	-0.17	-0.14	0.21	0.25	-0.07	-0.18	-0.27	1.00								
(9) Flow 1Qtr lag1	-0.18	-0.15	0.35	0.32	-0.09	-0.17	-0.33	0.81	1.00							
(10) Flow 1Yr lag1	-0.20	-0.16	0.25	0.40	-0.11	-0.12	-0.39	0.63	0.83	1.00						
(11) Turnover Ratio	-0.12	-0.15	-0.16	-0.21	-0.44	0.49	-0.29	-0.11	-0.11	-0.08	1.00					
(12) ICI lag2	-0.17	-0.16	-0.16	-0.01	-0.30	0.33	-0.10	-0.19	-0.21	-0.17	0.14	1.00				
(13) R2 Selectivity	-0.21	-0.19	-0.13	0.03	-0.37	0.46	-0.20	-0.16	-0.15	-0.09	0.23	0.84	1.00			
(14) Ret Gap lag2	-0.10	-0.08	-0.25	-0.08	-0.06	-0.11	-0.07	-0.19	-0.19	-0.19	-0.10	-0.15	-0.20	1.00		
(15) Ret Gap lag2 12M lag2	-0.08	-0.09	-0.36	0.06	-0.10	-0.07	-0.13	-0.15	-0.18	-0.16	-0.07	-0.12	-0.21	0.50	1.00	
(16) Active Share lag2	-0.13	-0.15	-0.05	-0.06	-0.52	0.60	-0.29	-0.17	-0.16	-0.11	0.27	0.42	0.53	-0.16	-0.14	1.00

1.5 Empirical Evidence

In this section, I present the empirical results. First, I will show the influence of the number of non-linear features and noise-to-signal ratio on the predicting power of machine learning methods. Then using machine learning models, I construct quintile portfolios based on the predicted fund returns, and test whether a hypothetical long-short trading strategy can earn risk-adjusted returns, thereby adding new evidence from the angle of machine learning on the debate whether mutual fund managers have skills. It turns out that machine learning methods don't have predicting power higher than OLS, in turn suggesting there is no-skill on average among mutual fund managers. And in the next subsection I will dive deeper to examine the exact reason of the failure. Finally, the same methodology is applied to hedge fund data and the results are presented.

1.4.1 How Does the Number of Non-linear Features and Noise-to-signal Ratio Affect the Performance of Machine Learning Methods? A Simulation

The current successful machine learning applications are based on relatively clean data that are less noisy, in terms of both features and labels. For example, given pictures of animals, predict the species of the animal. In this case, the picture is clean, and the label is 100% accurate. Also, machine learning methods are generally good at uncovering non-linear relationships based on big data. Here, "big" means both the number of features and the number of observations. However, it is well-known that financial data is highly noisy. A regression R^2 of 5% of a prediction for returns is not abnormal. Therefore, before applying machine learning methods to financial data and drawing any conclusions, I want to run a pilot test on simulated data to get a rough idea of how machine learning methods will perform under different conditions. Also, the simulation will act as a first-step screening: only the best performing methods will be applied to real data.

1.4.1.1 Simulation Assumptions

There are four assumptions underlying the simulation:

1) The joint distribution of features and labels are time invariant.
2) The measurement error only presents in features.
3) The relationship between features and labels are non-linear.
4) There are no omitted variables.

1.4.1.2 Data Generating Process

The data generating process is described as the following:

$$y = f(X_s),$$

where y is the label, $X_s \sim N(0, I)$ is the matrix of state variables that is not directly observable. $X_s \in R^{N \times M}$, where N is the number of observations and M is the number of non-linear features, and $f(\cdot)$ is a polynomial function of order 2 with interactions. What the economist can observe is $X = X_s + \varepsilon_X$, where $\varepsilon_X \sim N(0, \sigma_{\varepsilon_X}^2)$. Also, the noise-to-signal ratio is defined as $\frac{\sigma_{\varepsilon_X}}{\sigma_{X_s}}$. M is assumed to have one of the five levels from [5, 10, 20, 40, 80], and $\frac{\sigma_{\varepsilon_X}}{\sigma_{X_s}}$ is assumed to take one value from [0, 0.1, 0.5, 1, 10].

1.4.1.3 Machine Learning Model Performance on Simulated Data

To examine how the machine learning methods perform under different noise level and different number of non-linear features, according to the data generating process, for each level of M and $\frac{\sigma_{\varepsilon_X}}{\sigma_{X_s}}$, I generated 25 samples, each of them containing 450000 observations, which is about the same number of observations in the mutual fund data. Then each sample is randomly split into a training set of 400000 observations which is used to fit models, a validation set of 40000 observations which is used to tune hyper-parameters, and a test set of 10000 observations which is used to evaluate model performance. The machine learning models are trained and tuned on each of the 25 simulated data sample according to the procedures described in section 1.3.2. The model performances are presented in the following graphs.

Figure 1.1a shows the performance of Ordinary Least Square under various levels of noise-to-signal ratio and number of non-linear features. The x-axis represents five levels of noise-to-signal ratio $\frac{\sigma_{\varepsilon_X}}{\sigma_{X_S}}$ from 0 to 10, the y-axis represents five levels of the number of non-linear features M from 5 to 80, and the z-axis is performance, which is measured by out-of-sample R^2. From the simulation results of OLS we can see that when noise-to-signal ratio is low and there is no non-linear feature, OLS performs best, the out-of-sample R^2 can be as high as 95%. However, as the noise-to-signal value increases and with the presence of non-linear features, the performance deteriorates quickly. This result is expected since OLS only models linear relationships.

Figure 1.1b to Figure 1.1h show the performance advantage of machine learning models over OLS, including LASSO, Ridge, principal component analysis (PCA), partial least squares (PLS), random forest (RF), gradient boost random trees (GBRT), and neural networks (NN), respectively. The x-axis and y-axis are the same as Figure 1.1a, the z-axis is the performance advantage of the machine learning model over OLS. The performance advantage means the outperformance of a particular machine learning model over OLS.

Relatively higher values of performance advantage are in red colors and lower values are in blue colors.

From Figure 1.1b and Figure 1.1c, we can see that LASSO and Ridge perform basically the same as OLS, though under lower values of noise-to-signal ratio and number of non-linear features, they are a marginal advantage over OLS. Figure 1.1d shows the performance advantage of PCR over OLS. Unfortunately, PCR does even worse than OLS under all levels of noise-to-signal ratio and number of non-linear features. PLS is better than PCR, as shown in Figure 1.1e, however it has only marginal advantage over OLS under high levels of number of non-linear features. The results from Figure 1.1b to Figure 1.1e suggest that linear methods are not capable of capturing the underlying relationship in the data.

Figure 1.1f and Figure 1.1g show the performance advantage of random forest and gradient boost random tree respectively over OLS. We can see that under most circumstances, tree-based methods provide performance advantage over OLS. And they are significantly better than OLS at moderate level of number of non-linear features, and the advantage could be as high as 25% in terms of out-of-sample R^2. Furthermore, neural network performs significantly better than OLS even when the number of non-linear features is high, as shown in Figure 1.1h. Also, NN is reasonably resistant to noise, it has a performance advantage of 20% even at level 4 noise-to-signal ratio.

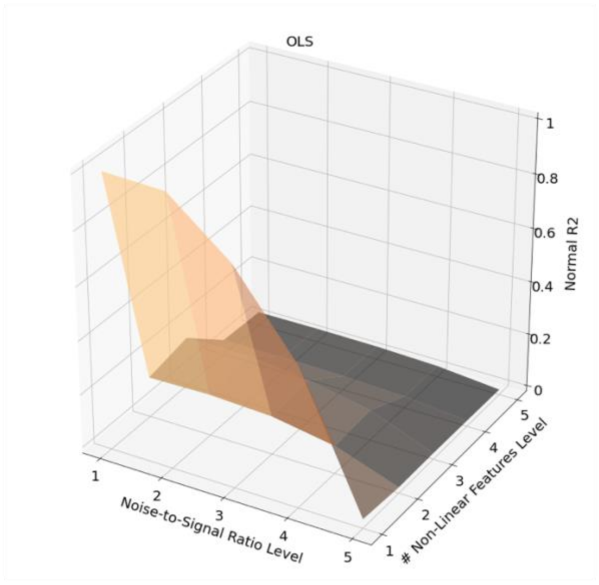

Figure 1.1a OLS Performance on Simulated Data

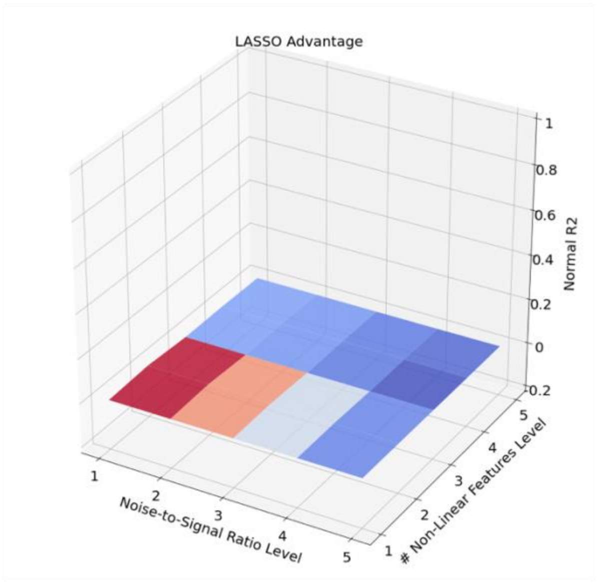

Figure 1.1b LASSO Performance Advantage over OLS on Simulated Data

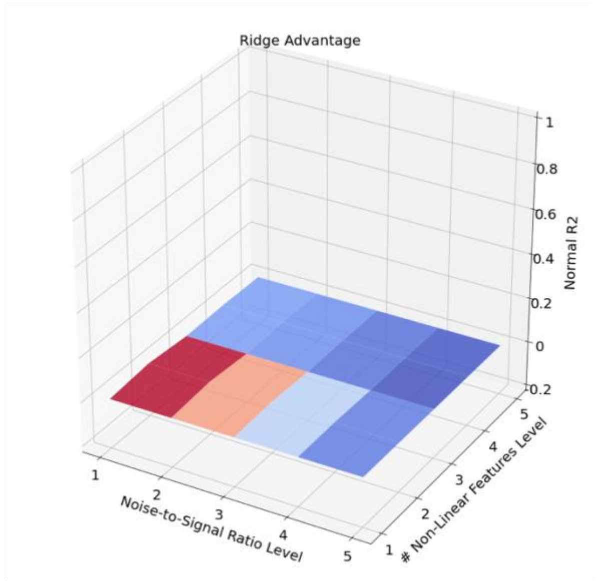

Figure 1.1c Ridge Performance Advantage over OLS on Simulated Data

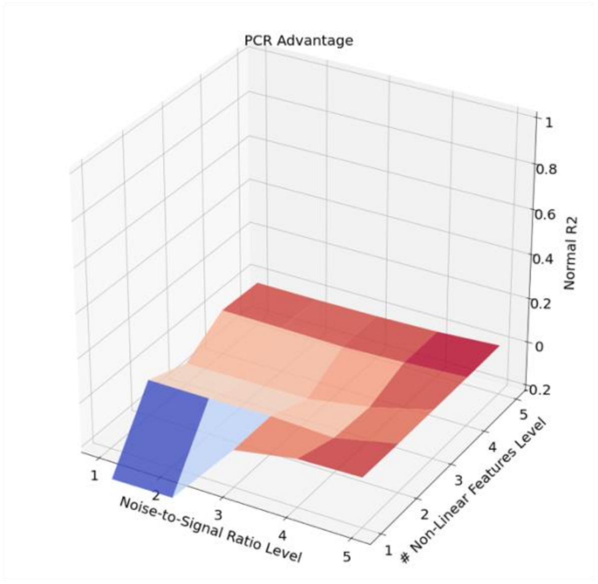

Figure 1.1d PCR Performance Advantage over OLS on Simulated Data

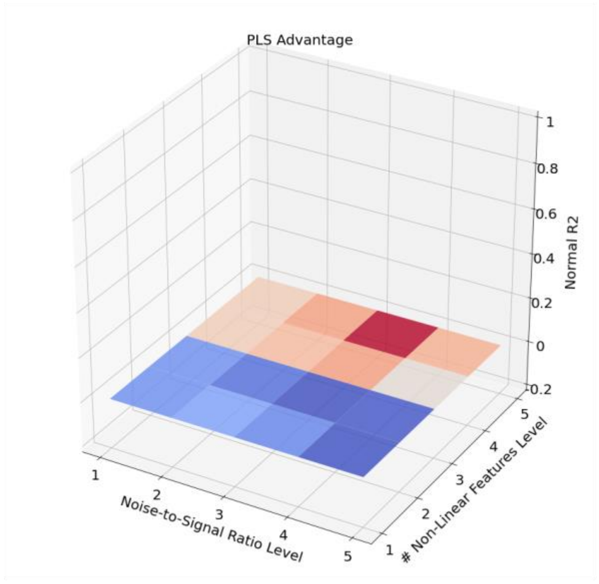

Figure 1.1e PLS Performance Advantage over OLS on Simulated Data

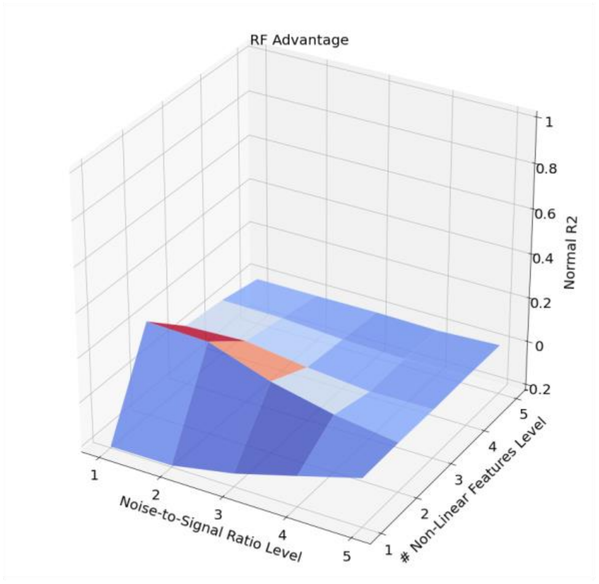

Figure 1.1f RF Performance Advantage over OLS on Simulated Data

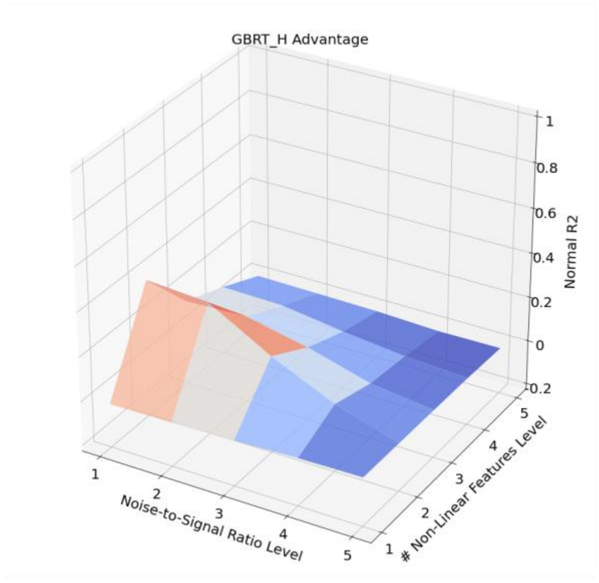

Figure 1.1g GBRT Performance Advantage over OLS on Simulated Data

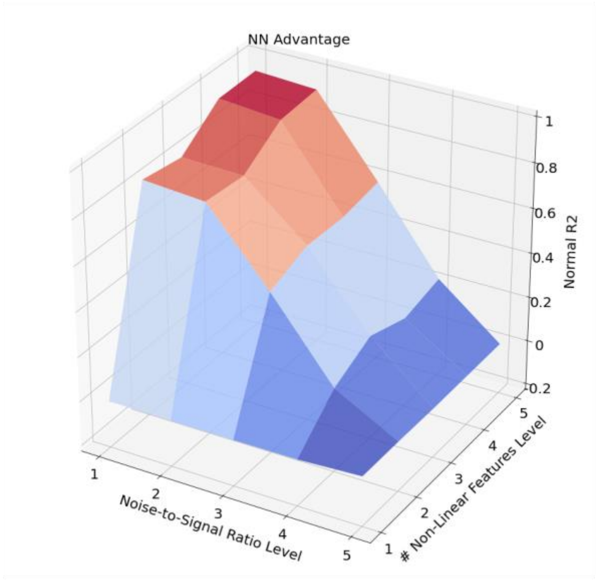

Figure 1.1h NN Performance Advantage over OLS on Simulated Data

To summarize the simulation result, the linear-based models don't yield satisfying results. On the contrary, non-linear models, i.e., RF, GBRT, and NN provide much better results over OLS. Therefore, in the tests that follows, I will only use these three models. Another important observation from the simulation is that, given the NN outperforms OLS most of the cases, the only cases where NN does not yield better performance over OLS is when either noise-to-signal ratio is extremely high, or number of non-linear features are very low.

1.4.2 Do Mutual Fund Managers Have Skill? Empirical Evidence from Machine Learning Methods

In this subsection, I will apply the three selected machine learning methods based on results from simulation, together with OLS as a benchmark, on mutual fund data.

1.4.2.1 Machine Learning Model Performance V.S. OLS Performance

Following the back-testing methodology described in Section 1.3.2, I train and tune each model as the following. The mutual fund data sample spans from 1983 to 2017. In order to train, tune, and evaluate the models, and to avoid introducing look-ahead bias, I use an expanding window of data starting from 1983 as training set and a fixed size window of 1-year data as test set. For example, to evaluate the out-of-sample performance of a model on data from 1998, the model is trained and tuned with 3-fold cross-validation on data from 1983 to 1997. And to evaluate the model on 1999, the model would be trained and tuned on data from 1983 to 1998, and so on so forth. In this way, each model is recalibrated during each year from 1998 to 2017. Using the model obtained, the predicted return of each mutual fund for each month of a particular year is computed. And the predicted returns of all years are stacked vertically with respect to time and side by side with realized returns to calculate out-of-sample model performance.

The results of model performance are presented in Table 1.3, which shows the out-of-sample R^2 of the three machine learning models with OLS as the benchmark. Table 1.3 shows both the performance of each model in each individual year from 1998 to 2017 and the overall performance of the whole sample period. From Table 1.3 we can see that OLS does a poor job in predicting mutual fund returns. The overall out-of-sample R^2 of OLS is -3.98%, which suggests that OLS is dominated by a naïve forecasting model that predicts zero return at all times. On the contrary, the machine learning models perform better than OLS. Specifically, random forest yields a higher GKX R^2 of 0.29%, gradient boost random trees yield a higher GKX R^2 of 0.4%, and neural network performs best, it yields the highest GKX R^2 of 0.52%. Also, by examining year by year performance, RF and GBRT perform better than OLS in 17 out of 20 years, and NN performs better than OLS in 19 out of 20 years. The results obtained from real data are consistent with the results in simulation. In summary, we do see a performance boost by using machine learning models.

1.4.2.2 Machine Learning Portfolio Performance

Now that I showed machine learning models are more effective predicting mutual fund returns, I want to address on the long-debated topic: whether mutual fund managers have skills. On one hand, Fama (1970) introduces the famous efficient market hypothesis, citing evidences that active mutual funds under-perform the market. Carhart (1997) argues that mutual fund managers do not have skills by using net alpha earned by investors. Zheng (1999) argues that there is no significant evidence that funds receive more money subsequently beat the market, suggesting that there is no performance persistence. On the other hand, a part of the literature does find evidence of skill. Grinblatt and Titman (1989) find positive gross alpha for small and growth funds. Wermers (2000) finds stocks that are held by mutual funds outperform market indices and argues that funds pick stock well enough to cover their costs. Kacperczyk, Sialm, and Zheng (2008) find that unobserved actions of some funds add value, suggesting that these funds do have skills.

However, no matter which side these papers take, the tool used in these papers are simple OLS. It's possible that it is the incapability of OLS that prevents those papers from discovering evidence for skill. Machine learning methods, as shown in the previous sections both with simulation and real data, are more powerful than OLS. Therefore, I will revisit the debate with the help of these more powerful methods.

To assess the skill of mutual fund managers, I use a zero-investment strategy to test whether this strategy can achieve positive risk-adjusted return. Specifically, at the end of each month from 1998 to 2017, I sort all funds into five quintiles Q1 to Q5 by the predicted return of the next month given by OLS and machine learning methods, with Q1 being the lowest predicted return quintile group and Q5 being the highest. Then for each quintile group in each month, a simple average of the realized return across all funds is calculated, representing the return of a strategy to invest equally in each mutual fund in the quintile group. Next, the return of the zero-investment strategy is calculated as the return of Q5 minus the return of Q1, meaning long Q5 and short Q1. Thus, we will have a total of six return time series.

Table 1.3 Out-of-Sample Performance of Machine Learning Models

Year	Model			
	OLS	RF	GBRT	NN
1998	6.28%	8.94%	4.96%	7.62%
1999	7.94%	11.16%	12.72%	11.79%
2000	-5.73%	-0.48%	0.77%	2.01%
2001	-24.77%	-7.64%	-7.93%	-8.22%
2002	-7.67%	-7.79%	-6.17%	-4.57%
2003	5.96%	6.45%	7.28%	8.10%
2004	0.63%	3.43%	5.13%	5.69%
2005	1.26%	3.29%	2.69%	2.99%
2006	8.62%	9.90%	9.30%	10.50%
2007	1.28%	6.39%	5.80%	6.97%
2008	-9.02%	-6.57%	-6.31%	-5.55%
2009	-14.65%	-12.02%	-11.72%	-13.50%
2010	2.90%	7.33%	7.02%	6.70%
2011	-6.04%	-0.63%	-0.99%	-2.44%
2012	5.14%	4.92%	5.43%	7.47%
2013	19.13%	21.63%	20.70%	21.16%
2014	0.45%	0.21%	-0.40%	1.40%
2015	-16.32%	-2.03%	0.50%	-1.52%
2016	7.45%	12.41%	11.96%	7.86%
2017	19.41%	20.77%	18.64%	17.93%
Overall	-3.98%	0.29%	0.40%	0.52%

Panel A in Table 1.4 shows the average raw return of the quintile portfolios of each model. We can see that for each model, the average realized return of the quintile portfolios increases monotonically, indicating that machine learning models are providing consistent return predictions with realized returns. Furthermore, for zero-investment portfolios, OLS earns an average realized return of 0.19% per month, RF earns 0.30%, GBRT earns 0.32%, and NN earns 0.41%. Machine learning models do earn a better return on the zero-investment strategy. This result is consistent with the previous findings that machine learning models are more versatile in discovering non-linear relationships. It should be noted that the zero-investment return for OLS, RF and GBRT are not statistically significant. However, NN does provide a statistically significant raw return.

It appears that neural network model is promising to show that mutual fund managers are skillful. However, it is possible that the zero-investment portfolio constructed by neural networks takes excess risk such that it shows non-zero returns. Therefore, in Panel B, Panel C, and Panel D, I use Capital Asset Pricing Model (CAPM, Lintner (1965)), Fama-French 3-Factor Model (Fama and French (1992, 1993)), and Carhart 4-Factor Model (Carhart (1997)), respectively, to calculate the risk-adjusted returns for the quintile portfolios and the zero-investment portfolio.

Panel B in Table 1.4 shows that compared with Panel A results, the quintile portfolios earn much less CAPM alpha in terms of magnitude. Also, unlike in Panel A, in which the raw returns are all statistically significant, there are a smaller number of quintile portfolios that have statistically significant CAPM alpha. For OLS, RF, and GBRT, the CAPM alpha is insignificant, which is not surprising. However, CAPM alpha for neural networks is marginally significant at 0.41%, which is the same as the raw return, suggesting that CAPM cannot explain the excess return earned by zero-investment portfolio constructed by NN. It appears that neural network provides some evidence that mutual fund managers are skillful. However, CAPM only adjusts risk for the market and ignore other risks. Thus, we turn to Fama-French 3-Factor Model, which adjusts for small capitalization and high book-to-market ratio.

Panel C shows the Fama-French alpha. We can see that there is an even smaller number of alphas that are significant for quintile portfolios, suggesting that most of the quintile portfolio returns can be explained by the three factors of Fama-French model. For the zero-investment portfolio of neural networks, Fama-French alpha is smaller than CAPM alpha at 0.31%, and it is still marginally significant.

Panel D shows the Carhart alpha. The Carhart alpha of zero-investment portfolios of OLS, RF, and GBRT are insignificant, and very close to 0 in magnitude. Also, the Carhart alpha for the zero-investment portfolio of neural network becomes insignificant at 0.20%, suggesting that the excess returns of the portfolio can be explained by market risk, small capitalization, high book-to-market ratio, and momentum factors.

In summary, machine learning models do provide higher raw return potentials than simple OLS. However, such higher return achieved by zero-investment portfolio can be explained by Carhart 4-Factor Model.

Table 1.4 Machine Learning Quintile Portfolio Performance

Panel A: Raw Return

Quintile	OLS	RF	GBRT	NN
Q1	0.96%***	0.72%***	0.94%***	0.55%*
Q2	1.02%***	0.85%***	1.04%***	0.66%**
Q3	1.03%***	0.98%***	1.05%***	0.69%**
Q4	1.09%***	1.01%***	1.26%***	0.78%**
Q5	1.16%***	1.02%***	1.26%***	0.96%***
Q5-Q1	0.19%	0.30%	0.32%	0.41%**

Panel B: CAPM Alpha

Quintile	OLS	RF	GBRT	NN
Q1	0.11%	0.10%	0.15%	0.00%
Q2	0.22%**	0.23%**	0.26%**	0.12%*
Q3	0.25%**	0.36%***	0.22%*	0.15%*
Q4	0.33%**	0.37%**	0.43%**	0.25%***
Q5	0.40%*	0.36%*	0.41%**	0.41%***
Q5-Q1	0.29%	0.27%	0.25%	0.41%**

Panel C: Fama-French Alpha

Quintile	OLS	RF	GBRT	NN
Q1	0.10%	0.11%	0.12%	0.01%
Q2	0.17%	0.19%**	0.20%**	0.10%
Q3	0.17%	0.27%**	0.15%	0.12%
Q4	0.22%	0.24%*	0.32%**	0.19%**
Q5	0.24%	0.20%	0.26%*	0.31%***
Q5-Q1	0.14%	0.10%	0.13%	0.31%**

Panel D: Carhart Alpha

Quintile	OLS	RF	GBRT	NN
Q1	0.15%	0.17%*	0.17%*	0.05%
Q2	0.19%*	0.20%**	0.20%**	0.12%*
Q3	0.19%*	0.24%**	0.16%	0.12%
Q4	0.21%	0.19%	0.30%**	0.18%**
Q5	0.20%	0.14%	0.23%	0.24%**
Q5-Q1	0.05%	-0.03%	0.06%	0.20%

$* \, p \leq 0.05$, $** \, p \leq 0.01$, $*** \, p \leq 0.001$

1.6 Summary

In this chapter, I use three machine learning models, random forest, gradient boosting random trees, and neural networks, to predict mutual fund returns with a comprehensive data with 94 independent variables. Zero-investment portfolios constructed by taking long positions in the best quintile of funds and short positions in the worst quintile of funds based on machine learning models are able to provide higher raw return than OLS. However, such outperformance is explained by Carhart 4-factor model. Machine learning models are best at uncovering non-linear relationships with high volume of data. Such result suggests that even with superior tool of machine learning, we cannot construct a zero-investment portfolio with positive risk-adjusted returns, which implies that the mutual fund industry on average does not provide positive risk-adjusted returns.

To my knowledge, this is the first research to apply machine learning methods in mutual fund performance prediction, and it adds an extra piece of evidence on the lack of skills of mutual funds managers to the mutual fund literature from an angle of machine learning.

Chapter 2

On the Skills of Hedge Fund Managers via Machine Learning Methods

2.1 Background

In the previous chapter, I use three machine learning models, random forest, gradient boost random trees, and neural networks, to predict the performance of mutual funds[3]. Although machine learning methods are promising to provide higher raw returns with a zero-investment portfolio by taking long positions on those funds that are predicted as best performers and short the worst ones, such portfolio does not provide statistically significant performance after adjusting for risks. Such results suggest that mutual fund managers on average do not add value.

However, it is well documented in hedge fund literature that at least a portion of hedge fund managers do deliver positive, statistically significant risk-adjusted returns. For example, Ackermann, McEnally, and Ravenscraft (1999) show that hedge funds have both higher risk-adjusted performance and higher levels of risk, and outperform equities and fixed income asset classes. Liang (1999) shows that hedge funds provide higher Sharpe ratios than mutual funds, and the performance reflects better manager skills. Fung and Hsieh (2005) show that equity long-short hedge funds have significant alpha to both conventional as well as alternative risk factors.

Furthermore, as another way to distinguish managerial skill from luck, many authors examined the persistence of hedge funds' abnormal performance. For instance, Agarwal and Naik (2000) argue that hedge funds do show performance persistence, the persistence is at its maximum at quarterly horizon, and it decreases at yearly horizon. Edwards and Caglayan (2001) find that hedge funds exhibit persistence over one- and two-year horizons and that both winners and losers exhibit significant persistence. Koh, Koh,

[3] Readers are encouraged to read Section 1.2 for the introduction of machine learning models used in this chapter.

and Teo (2003) find that Asian hedge funds exhibit persistence most strongly at monthly horizons to quarterly horizons. Baquero, Horst, and Verbeek (2005) find positive persistence in hedge fund quarterly returns after correcting for investment style. Also, Jagnnathan, Malakhov, and Novikov (2010) examine whether "hot hands" exist among hedge fund managers. The authors use both relative performance and absolute performance to rank funds and find that portfolios constructed by both ranking methods exhibit persistence at a three-year horizon and such persistence mainly exists with top performers.

In this chapter, I will continue to explore the possibility to predict hedge fund performances with the three machine learning methods used in the previous chapter. The research questions that are pursued include the following. For example, are portfolios constructed by machine learning methods be able to achieve positive risk-adjusted returns? And if so, is the performance of the portfolios persistent? How does the performance of machine learning models compare with OLS? What factors or variables could potentially drive the outperformance? And in what circumstances does the outperformance tend to occur?

The rest of this chapter is organized as follows. Section 2.2 describes the data. Section 2.3 presents the methodology of hedge fund performance evaluation and the methodology to evaluate performance persistence. Section 2.4 presents empirical results. And Section 2.5 summarizes the findings.

2.2 Data

In order to examine the performance of the machine learning models on hedge fund returns, I use Lipper Trading Advisor Selection System (TASS) database. TASS is one of the few vendors that provide hedge fund data, which includes variables such as the monthly returns, fund style, fee structures, number of lockup months, whether a personal capital is dedicated, age of funds, etc. Due to the facts that hedge funds are less regulated and their trading strategies are highly secretive, they are not required to report their holdings. Therefore, TASS does not have holdings information unfortunately.

One known bias existing in hedge fund databases is backfill bias, which is a variation of survivorship bias. This happens when hedge fund databases allow funds to back fill their past performance. Such behavior causes an upward bias: the performance of hedge funds could be inflated. Since hedge fund reporting is voluntary, those funds who are outperformers tend to join hedge fund databases for marketing purposes, and those funds that are underperformers are not incentivized to report their performances. Therefore, removing backfill bias is an essential step in conducting hedge fund research.

To correct backfill bias, I deleted all the backfilled observations for which the date of return is earlier than when the fund actually joined the database. My final sample spans from 1997 to 2015, which includes a total of 731362 observations. There are 15079 distinct hedge funds in the sample, and the average monthly raw return is 0.27%.

2.3 Methodology

2.3.1 Performance Evaluation Models

In this chapter, I use three methods to evaluate the performance of a hedge fund or a portfolio. The first method is Fung and Hsieh seven-factor model Fung and Hsieh (2004)[4] as the following:

$$r_{i,t} = \alpha_i + \beta_{i,1}PTFS_{B_t} + \beta_{i,2}PTFS_{Cur_t} + \beta_{i,3}PTFS_{Com_t} + \beta_{i,4}EM_t + \beta_{i,5}SS_t + \beta_{i,6}BM_t + \beta_{i,7}CS_t + \epsilon_{i,t},$$

where $PTFS_{B_t}$, $PTFS_{Cur_t}$, and $PTFS_{Com_t}$ are trend-following factors for bond, currency, and commodity, respectively. EM_t is equity market factor, SS_t is size spread factor, BM_t is bond market factor, and CS_t is credit spread factor. For each fund, the return series is regressed on the above seven factors. The intercept of the regression is the risk-adjusted performance of the fund.

The second method is machine learning alpha. For each fund, a particular machine learning model, such as random forest, gradient boost random tree, or neural network, is fitted by taking Fung and Hsieh 7 factors as the input and the fund's return series as the output. Then, the predicted return from the machine learning model is calculated and a series of residual between the realized returns and the predicted returns is obtained. Finally, the machine learning alpha is obtained by taking the time series average of the residuals.

The third model employed is style-adjusted alpha. In particular, hedge funds are first grouped into portfolios by investment styles. Next, the monthly returns of each style

[4] https://faculty.fuqua.duke.edu/~dah7/HFData.htm

portfolio are calculated as equal-weighted mean of the return of the underlying hedge funds. Finally, to obtain the style-adjusted alpha, the return series of each fund is regressed on its corresponding style portfolio return series. The regression intercept is the style-adjusted alpha.

2.3.2 Persistence Evaluation

I employ the methodology in Jagannathan, Malakhov, and Novikov (2010) to evaluate performance persistence. In particular, hedge fund alphas are compared over two consecutive non-overlapping three-year periods. During the first three-year period, which is also called the portfolio formation period, hedge funds are ranked by performance measures, such as Fung and Hsieh 7-factor alpha or machine learning model alpha. Three mutually exclusive and collectively exhaustive portfolios are then constructed, with the Superior group containing hedge funds with the best 10% of funds, the Inferior group containing hedge funds with the worst 10% of funds, and the Neutral group containing the remaining funds. Then the risk-adjusted returns of the three performance groups are calculated in the formation period. Finally, the risk-adjusted returns of the three groups during the second three-year period, which is also called the testing period, are calculated. For a particular performance group, if the alphas from the portfolio formation period and the testing period are both of the same sign and statistically significant, the performance of such group is persistent.

2.4 Empirical Results

The existing literature has generally accepted that hedge funds deliver positive excess returns. However, whether hedge funds show performance persistence remains unclear. In this section, I first present the result of performance persistence from three machine learning portfolios as wells as OLS model and show that the persistence obtained from machine learning portfolios are potentially higher than a portfolio obtained by traditional OLS method.

2.4.1 Performance Persistence of Machine Learning Portfolios

To evaluate performance, I employ method described in Section 2.3.2. I use samples from rolling window of 6 years from 1997 to 2015, with the first three-year period as portfolio formation period, and the second three-year period as testing period. For OLS method, hedge funds are sorted into three portfolios by Fung and Hsieh 7-factor alpha during the portfolio formation period. For the machine learning method, hedge funds are sorted by machine learning alphas, for which the calculation procedure was described in Section 2.3.1. Once the portfolios are formed, Fung and Hsieh 7-factor alpha of both the formation period and testing period are calculated for all portfolios from each machine learning model, with OLS model as the benchmark. The results are shown in Table 2.1.

Looking at the results cross-sectionally, we can see that for OLS model, for each rolling window sample, the formation period alphas are strictly monotonically increasing, which is not surprising since by construction the hedge funds are sorted by alphas. Also, for machine learning models, the formation period alphas are almost monotonically increasing, suggesting that machine learning models evaluate the performance of hedge funds in a similar way to OLS. For testing period results, all models send the same message in terms of performance persistence: the Superior portfolios show persistence most of the time (9 out of 14 rolling window samples) while the Inferior portfolios don't. This result is qualitatively similar to Jagannthan, Malakhov, and Novikov (2010), which finds significant performance persistence among superior funds but little evidence of persistence among inferior funds.

Looking at the results time-seriesly, we can see that not all of the rolling window samples show performance persistence. For example, the sample 1997-1999 to 2000-2002, and samples with testing period from 2008 to 2011. The Superior portfolios do not show persistence during these years, possibly because of the tech bubble crisis and the 2008 sub-prime crisis. More tests will be focused on the association of machine learning model performance and economic hardships in the following sections.

Comparing machine learning models with their benchmark OLS model, we can see that the Superior portfolios formed by machine learning methods provide higher testing period alphas than OLS model, while the out-of-sample performance of Inferior portfolios are similar among machine learning models and OLS model. Among the machine learning models, neural network provides the best out-of-sample performance. This result suggests that machine learning methods may be better at evaluating hedge fund performances and that portfolios based on machine learning methods may provide a higher out-of-sample risk-adjusted return for investors.

Table 2.1 Performance Persistence of Machine Learning Models

		OLS		RF		GBRT		NN	
Cross-Section	Group	Formation Period Alpha	Testing Period Alpha	Formation Period Alpha	Testing Period Alpha	Formation Period Alpha	Testing Period Alpha	Formation Period Alpha	Testing Period Alpha
1997-1999 to 2000-2002	Inferior	-1.47***	-0.01	-0.06	-0.10	-0.99***	-0.30	-0.30**	-0.70
	Neutral	1.57***	-0.33	1.48***	-0.32	1.53***	-0.33	1.27***	-0.29
	Superior	2.02***	0.03	1.19***	-0.10	1.88***	0.09	2.37***	0.00
1998-2000 to 2001-2003	Inferior	-1.11***	0.10	0.02	-0.03	-0.93**	0.19	0.39	0.07
	Neutral	0.88*	0.10	0.68	0.07	0.86*	0.08	0.37**	0.18
	Superior	1.17***	0.25***	1.34**	0.54***	1.21***	0.27***	0.78***	0.32***
1999-2001 to 2002-2004	Inferior	-0.96***	0.07	-0.58	0.36	-0.92***	-0.30	-0.10	0.13
	Neutral	0.65	0.15	0.48	0.10	0.65	0.18	0.48	0.34
	Superior	0.92***	0.40***	0.44**	0.46***	0.93***	0.43***	1.00***	0.35***
2000-2002 to 2003-2005	Inferior	-1.71***	0.50	-0.25	0.24	-0.07	0.24	-1.56***	0.21
	Neutral	0.09	0.44**	-0.03	0.42**	0.07	0.44**	0.03	0.35*
	Superior	0.81***	0.27***	0.29***	0.73**	0.26**	0.59***	1.17***	0.73***
2001-2003 to 2004-2006	Inferior	-0.85***	0.40	0.09	0.33	-0.13	0.39	-0.02	0.32
	Neutral	0.30	0.64***	0.24	0.64***	0.23	0.62***	0.22	0.58***
	Superior	0.79***	0.53***	0.34***	0.43**	0.63***	0.64***	0.65***	0.95***
2002-2004 to 2005-2007	Inferior	-0.71**	0.31	0.16	0.21	0.06	0.27	0.18	0.27
	Neutral	0.42**	0.59***	0.34	0.55	0.35*	0.55**	0.27	0.48***
	Superior	0.88***	0.45***	0.58***	0.61***	0.62***	0.64***	1.15***	1.19***
2003-2005 to 2006-2008	Inferior	-0.55***	0.11	-0.41	0.51	0.29	0.22	0.13	0.30
	Neutral	0.50**	0.35	0.23	0.27	0.40*	0.29	0.36**	0.23
	Superior	0.98***	-0.04	0.47**	0.18	0.91***	0.28	0.96***	0.28

$* \, p \leq 0.05$, $** \, p \leq 0.01$, $*** \, p \leq 0.001$

Table 2.1 Performance Persistence of Machine Learning Models - Continued

		OLS		RF		GBRT		NN	
Cross-Section	Group	Formation Period Alpha	Testing Period Alpha	Formation Period Alpha	Testing Period Alpha	Formation Period Alpha	Testing Period Alpha	Formation Period Alpha	Testing Period Alpha
2004-2006 to 2007-2009	Inferior	-0.41**	0.13	0.62***	0.09	0.62**	-0.21	0.55	-0.05
	Neutral	0.71***	0.03	0.53**	0.02	0.60***	0.03	0.75***	0.32
	Superior	1.06***	-0.38	0.77***	-0.22	0.87***	0.04	1.12***	-0.01
2005-2007 to 2008-2010	Inferior	-0.35***	-0.13	0.59***	-0.29	0.42*	-0.32	0.08	-0.13
	Neutral	0.61***	-0.02	0.56**	-0.02	0.55*	-0.01	0.51***	-0.10
	Superior	1.06***	-0.05	0.53***	0.17	0.77***	0.12	1.47***	0.48
2006-2008 to 2009-2011	Inferior	-0.72***	-0.04	-0.30	0.12	0.08	0.23	0.35	0.20
	Neutral	0.42	0.23	0.39	0.23	0.38	0.20	0.40	0.15
	Superior	1.18***	0.30	0.44	0.23	0.71***	0.31	0.55	0.27
2007-2009 to 2010-2012	Inferior	-0.99***	0.05	-0.97***	-0.05	-0.46	0.02	0.09	0.18
	Neutral	0.17	0.24	0.18	0.25	0.15	0.25	0.31*	0.54
	Superior	1.21***	0.39***	1.16***	0.37***	0.88***	0.42*	0.63**	0.44**
2008-2010 to 2011-2013	Inferior	-1.31***	0.11	-1.08***	-0.10	-0.55	0.53	-0.42	0.19
	Neutral	0.09	0.31	0.08	0.31	0.99**	0.42	0.03	0.26
	Superior	1.27***	0.34**	1.10***	0.40***	1.38***	0.43***	0.86***	0.49***
2009-2011 to 2012-2014	Inferior	-0.90***	0.26	-0.83***	0.35	-0.26	0.21	-0.21	0.12
	Neutral	0.22	0.41**	0.22	0.40**	0.20	0.41**	0.17	0.43***
	Superior	1.01***	0.39***	0.91***	0.46***	0.57***	0.48***	0.66***	0.44***
2010-2012 to 2013-2015	Inferior	-0.78***	0.02	0.19	0.29	-0.09	0.21	0.05	0.39
	Neutral	0.27	0.48***	0.25	0.46***	0.23	0.47***	0.18	0.56***
	Superior	0.96***	0.50***	0.08	0.49***	0.57***	0.53***	0.84***	0.72***

$* \ p \leq 0.05$, $** \ p \leq 0.01$, $*** \ p \leq 0.001$

Table 2.2 Style-Adjusted Alpha for Machine Learning Portfolios in Testing Period

Cross-Section	Group	OLS	RF	GBRT	NN	Cross-Section	Group	OLS	RF	GBRT	NN
1997-1999 to 2000-2002	Inferior	0.19	-0.11	-0.68**	-0.32	2004-2006 to 2007-2009	Inferior	0.08	-0.06	-0.12	-0.11
	Neutral	-0.57***	-0.54***	-0.54**	-0.41***		Neutral	-0.05	-0.04	-0.06	-0.06
	Superior	0.00	-0.14	0.07	0.03		Superior	-0.36	-0.30	-0.15	-0.07
1998-2000 to 2001-2003	Inferior	-0.14	-0.46	0.04	-0.20	2005-2007 to 2008-2010	Inferior	-0.09	-0.20	-0.27***	-0.20
	Neutral	-0.22	-0.21	0.21	0.03		Neutral	-0.11	-0.10	-0.09	-0.11
	Superior	0.12**	0.38***	0.15***	0.14**		Superior	-0.06	0.03	-0.02	0.03
1999-2001 to 2002-2004	Inferior	-0.15	0.02	-0.26*	-0.16	2006-2008 to 2009-2011	Inferior	-0.10	-0.10	-0.12	-0.26
	Neutral	-0.05	-0.07	-0.04	-0.04		Neutral	-0.12	-0.12	-0.10	-0.07
	Superior	0.44***	0.41***	0.45***	0.38***		Superior	0.04	0.03	-0.14	-0.21
2000-2002 to 2003-2005	Inferior	-0.12	0.04	0.11	0.22	2007-2009 to 2010-2012	Inferior	-0.22**	-0.32***	-0.16**	-0.15
	Neutral	0.26***	0.22***	0.23***	0.18***		Neutral	-0.24***	-0.25***	-0.23***	0.22**
	Superior	0.38***	0.57***	0.45***	0.76***		Superior	0.12*	0.21***	0.17**	0.18***
2001-2003 to 2004-2006	Inferior	-0.12	-0.26	-0.10	0.07	2008-2010 to 2011-2013	Inferior	-0.42	-0.38	-0.35	-0.13
	Neutral	0.06	0.11	0.09	0.12*		Neutral	-0.30***	-0.31***	-0.27	-0.21***
	Superior	0.37***	0.18*	0.42***	0.48***		Superior	0.13	0.22***	0.22***	0.26***
2002-2004 to 2005-2007	Inferior	-0.04	0.04	0.08	0.03	2009-2011 to 2012-2014	Inferior	-0.20	-0.23	-0.18	-0.33
	Neutral	0.14**	0.12*	0.12**	0.18***		Neutral	-0.13*	-0.13*	-0.11	-0.02
	Superior	0.24***	0.28***	0.21**	0.31***		Superior	0.17***	0.18***	0.21***	0.23***
2003-2005 to 2006-2008	Inferior	-0.04	0.02	-0.09	-0.04	2010-2012 to 2013-2015	Inferior	-0.42***	-0.27**	-0.18**	-0.22***
	Neutral	-0.04	-0.12	-0.07	-0.09		Neutral	0.00	0.00	0.00	0.08
	Superior	-0.43*	0.02	-0.20	-0.13		Superior	0.26***	0.27**	0.20*	0.32***

$* \; p \le 0.05$, $** \; p \le 0.01$, $*** \; p \le 0.001$

It should be noted that in OLS regression one of the main assumptions is error terms are uncorrelated. Violation of this assumption may yield spuriously high value of alpha. This is called correlated estimation error. Therefore, the previous result of performance persistence might be due to the violation of the assumption of uncorrelated error. To address this concern, a simple cure is to use another method, such as style-adjusted alpha, to evaluate the out-of-sample portfolio performance. The out-of-sample style-adjusted alphas are shown in Table 2.2. The results are similar to that with FH 7-factor alpha.

2.4.2 Full Sample Investment Performance of Machine Learning Portfolios

In the previous subsection, I showed that machine learning portfolios provide higher performance persistence than OLS model with rolling window subsamples. In this subsection, I use the full sample data and create trading strategies according to OLS and machine learning models by investing in the Superior group funds and Inferior group funds, respectively, with the portfolios rebalanced every three years. Performance characteristics such as maximum draw down, max 1-month loss, Sharpe ratio and portfolio turnover ratio are reported. The raw return and Fung and Hsieh 7-factor adjusted performance of the portfolios are also presented, as well as the factor loadings. The results are shown in Table 2.3. The first four columns show the results of OLS and machine learning models, the last three columns show the difference between machine learning models and OLS model.

Panel A of Table 2.3 presents the performance characteristics of Superior portfolios. The three machine learning model portfolios have lower maximum draw down than OLS, which is at -25.47%. GBRT has the lowest maximum draw down of -13.54%. RF and NN has comparable maximum 1-month loss with OLS at roughly -7%. GBRT has the lowest 1-

month loss of -3.5%. For the Sharpe ratio, OLS and RF are less than 1, NN is slightly higher than 1, and GBRT is at 1.46. For the turnover ratio, all models are at the range of 15% to 20%. In terms of raw return, OLS has the lowest performance at 0.4% per month, RF is slightly higher at 0.42%, GBRT is at 0.5%, and NN has the highest performance at 0.55% per month. The raw performance difference is also tested. The raw performance difference between both GBRT and OLS and NN and OLS are statistically significant, while there is no statistically significant difference between RF and OLS.

Panel B presents the Fung and Hsieh 7-factor adjusted performance and factor loadings of the Superior portfolios. After risk adjustment, the alphas of all models are smaller than the raw performance but remain statistically significant. Also, the difference of alphas between GBRT and OLS and NN and OLS are statistically significant while there is no significance on the difference between RF and OLS. For NN model, it outperforms OLS in terms of risk-adjusted performance by 2.4% per year on average.

Panel C shows the performance characteristics for Inferior portfolios. The maximum draw down, max 1-month loss, Sharpe Ratio, and raw return of machine learning portfolios are all worse than OLS. However, the difference of raw returns between machine learning portfolios and OLS are not statistically significant.

Panel D shows the Fung and Hsieh 7-factor adjusted performance and factor loadings of the Inferior portfolios. After risk adjustment, the alphas of the portfolios are not statistically different from 0. And there is no significant difference of alphas between machine learning portfolios and OLS.

Table 2.3 Full Sample Machine Learning Investing Portfolio Analysis

	OLS	RF	GBRT	NN	RF - OLS	GBRT - OLS	NN - OLS
Panel A: Superior Group Performance Characteristics							
Max Draw Down (%)	-25.47	-21.79	-13.54	-22.53			
Max 1 Month Loss (%)	-7.09	-6.09	-3.50	-7.71			
Sharpe Ratio	0.96	0.97	1.46	1.09			
Turnover Ratio (%)	15.35	19.12	15.94	17.46			
Raw Return (% p.m.)	0.40***	0.42***	0.50***	0.55***	0.03	0.11**	0.16*
Panel B: Superior Group Fung and Hsieh 7 Factor Loadings							
Alpha (% p.m.)	0.17**	0.18*	0.30***	0.37***	0.01	0.13***	0.20**
S&P	5.77***	4.04**	2.50	2.48	-1.73	-3.28***	-3.29
SC-LC	11.08***	14.57***	12.12***	9.04**	3.49*	1.04	-2.04
10Y	-0.05	-0.73*	-0.40	-1.19**	-0.68***	-0.35*	-1.14***
CredSpr	-2.74***	-2.94***	-2.12***	-3.12***	-0.20	0.61**	-0.38
BdOpt	-2.18***	-2.32***	-1.15**	-0.77	-0.14	1.03***	1.41*
FXOpt	0.76	1.05*	1.12**	1.18	0.29	0.37	0.42
ComOpt	-0.24	0.30	0.53	1.79**	0.54	0.77**	2.03***
AdjR2 (%)	41.18	35.02	29.37	15.55	5.56	26.66	13.50
Panel C: Inferior Group Performance Characteristics							
Max Draw Down (%)	-16.31	-20.06	-23.25	-44.5			
Max 1 Month Loss (%)	-4.58	-6.51	-7.16	-11.98			
Sharpe Ratio	0.54	0.50	0.48	0.15			
Turnover Ratio (%)	26.08	23.43	22.66	21.99			
Raw Return (% p.m.)	0.25**	0.24**	0.22**	0.13	-0.01	-0.03	-0.13
Panel D: Inferior Group Fung and Hsieh 7 Factor Loadings							
Alpha (% p.m.)	0.03	0.08	0.02	-0.07	0.04	-0.01	-0.11
S&P	0.35	2.99	2.21	0.39	2.65	1.86	0.04
SC-LC	13.15***	11.54***	12.72***	32.45***	-1.61	-0.43	19.30***
10Y	-0.06	-0.23	-0.68*	-0.07	-0.17	-0.63*	-0.01
CredSpr	-2.74***	-4.17***	-4.25***	-3.48***	-1.43***	-1.51***	-0.74
BdOpt	-1.86**	-0.59	-1.30**	0.61	1.28**	0.57	2.47
FXOpt	1.12*	1.30**	0.91*	0.35	0.19	-0.21	-0.77
ComOpt	-1.12	-0.07	-0.42	1.58	1.05*	0.70	2.69*
AdjR2 (%)	29.44	38.70	46.21	18.38	7.88	5.18	3.43
Panel E: Superior Group Minus Inferior Group Risk-Adjusted Performance							
Alpha (% p.m.)	0.14	0.10	0.28***	0.45***	-0.04	0.15*	0.31**

$* \ p \leq 0.05$, $** \ p \leq 0.01$, $*** \ p \leq 0.001$

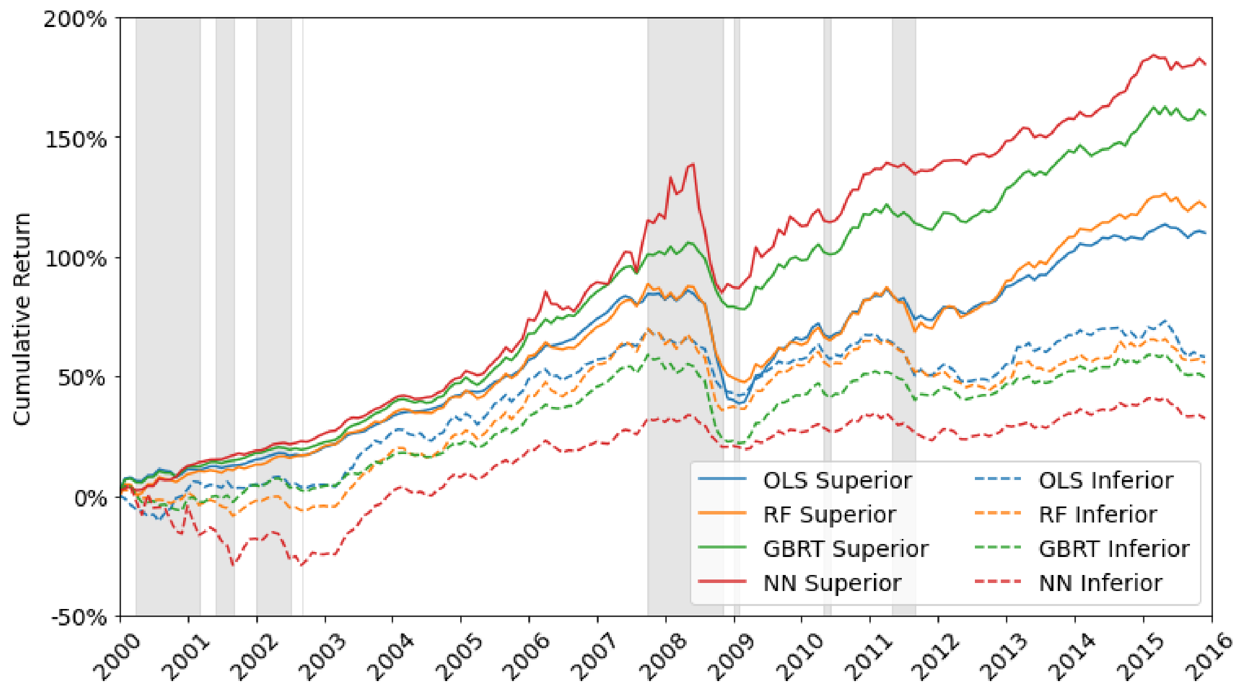

Figure 2.1 Machine Learning Portfolio Cumulative Performance

Panel E shows the risk-adjusted performance of a zero-investment trading strategy by taking long positions in hedge funds in the Superior portfolio and short positions in the Inferior portfolio. OLS and RF do not provide significant risk-adjusted performance while GBRT delivers significant monthly alpha at 0.28% and NN delivers significant monthly alpha at 0.45%.

The raw time series cumulative performance of both Superior portfolio and Inferior portfolio of each model are presented in Figure 2.1. The Superior portfolio performances are plotted in solid curves while the Inferior portfolio performances are plotted in dashed curves. It can be observed that OLS has the inner-most two curves, which suggests the ability of OLS to differentiate the best funds from the worst ones are at weakest. The machine learning models performance curves embraces the OLS curves. NN has the best Superior portfolio raw performance and the worst Inferior portfolio performance. And the ultimate difference between the Superior portfolio and the Inferior portfolio of NN are at its maximum at roughly 150%. The shaded grey areas indicate bear market periods. And it should be noted that during the 2008 financial crisis, all portfolios experienced heavy losses.

In summary, both Table 2.3 and Figure 2.1 suggest that machine learning models have higher ability to differentiate best funds from the worst ones, in terms of both raw return and risk-adjusted performance.

2.4.3 Robustness Check with Fund Characteristics as Control Variables

In this subsection I continue to explore the outperformance of machine learning models with a panel regression approach. The portfolio approach in previous subsections does not control hedge fund characteristics that are known to affect future performances. It is possible that the outperformances of machine learning models are driven by fund characteristics. To investigate this possibility, I use the following panel regression:

$$Abnormal\ Performance_{i,t} = const + \beta Control_{i,t-1} + \epsilon_{i,t}.$$

The abnormal performance is obtained by the following procedures. For each fund at each month, I first regress the previous 36-month returns on Fung and Hsieh 7 factors to obtain factor loadings. And the loadings are used in the current month to obtain the predicted return. The difference between the real return and the predicted return is treated as the abnormal performance of the current month.

For the control variables, I use 13 lagged fund characteristics, including performance volatility of past 2 years, measured in percent; the redemption notice period, measured in units of 30 days; lock up periods, measured in months; personal capital dummy; high water mark dummy; management fees and incentive fees, measured in percent; age in years; natural log of asset under management; flows into funds within the past 12 months as percentage of AUM; average monthly net-of-fee returns, measured in percent; natural log of minimum investment; and a leverage dummy as an indicator whether the fund uses leverage. In the panel regression, if the constant is statistically significant, it means that the abnormal performance cannot be explained by fund characteristics. Hence, the performance is not driven by fund characteristics.

The results of the panel regression are shown in Table 2.4. Panel A presents the panel regression using observations of hedge funds in the Superior portfolios. The constant of OLS and RF are not statistically different from 0 and the constant of GBRT and NN are positive and statistically significant. This demonstrates that the performance of OLS and RF may be driven by fund characteristics while the performance of GBRT and NN is not. GBRT and NN provide an average risk-adjusted return of 0.24% and 0.33%, respectively, even after controlling fund characteristics.

Panel B shows the result of the panel regression with observations of hedge funds in the Inferior portfolios. The constants of all models are not statistically different from 0, which is not surprising, since the abnormal performance of the Inferior group is not statistically significant either (see Panel D in Table 2.3).

In order to compare the relative effectiveness of machine learning models with OLS under the presence of control variables, I pool all the observations from both the Superior group and the Inferior group together, and run the following panel regression with an additional Superior group indicator, which takes value of 1 if the fund belongs to the Superior group and 0 otherwise:

Table 2.4 Panel Regression of Abnormal Performance on Control Variables

	OLS	RF	GBRT	NN
Panel A: Superior Group				
Const	-0.02	0.04	0.24***	0.33***
VolPast2Y (% p.m.)	-0.06***	-0.03***	-0.06***	-0.04***
RedemptionNotice (30Days)	0.00	-0.03***	-0.01	-0.03***
Lockup (months)	0.00***	0.00***	-0.01***	0.00***
PersonalCapitalDummy	0.06***	-0.01	0.01	0.01
HighWaterMarkDummy	-0.05***	-0.02*	-0.04***	-0.06***
MgmtFee (%)	0.06***	0.10***	0.02*	0.02*
IncentiveFee (%)	0.00***	0.00***	0.00***	0.00***
Age (years)	-0.01***	-0.01***	-0.02***	-0.02***
ln (AUM)	0.00	0.00	-0.01**	0.00
FlowPast1Y (%)	0.00***	0.00	0.00	0.00
AvgPast2YRet (% p.m.)	0.06***	0.02***	-0.02***	0.01**
Ln (MinInvestment+1)	0.02***	0.01***	0.02***	0.02***
Leverage	0.05***	-0.03***	0.08***	-0.03**
AdjR2 (%)	6.36	2.55	5.59	4.92
Panel B: Inferior Group				
Const	0.05	0.03	-0.03	-0.05
VolPast2Y (% p.m.)	-0.01***	-0.01***	-0.01***	-0.01
RedemptionNotice (30Days)	0.04***	0.01	0.08***	-0.04***
Lockup (months)	-0.01***	0.00***	0.00*	0.00***
PersonalCapitalDummy	-0.09***	-0.08***	-0.08***	-0.03***
HighWaterMarkDummy	0.04***	0.01	-0.01	0.07***
MgmtFee (%)	-0.09***	0.08***	0.08***	0.06**
IncentiveFee (%)	-0.01***	0.00***	0.00	0.00***
Age (years)	-0.02***	-0.01***	0.00	-0.02***
ln (AUM)	0.02***	0.00	0.00	0.01
FlowPast1Y (%)	0.00***	0.00***	0.00	0.00
AvgPast2YRet (% p.m.)	0.02***	-0.06***	0.00	-0.01**
Ln (MinInvestment+1)	-0.02***	-0.01	-0.03***	0.01
Leverage	0.07***	0.08***	0.01	0.04
AdjR2 (%)	4.40	2.51	3.54	2.76
Panel C: Both Superior and Inferior Group with Superior Group Indicator				
SuperiorGroupInd.	0.00	0.02*	0.21***	0.29***
Const	0.00	0.02	-0.02	-0.06
VolPast2Y (% p.m.)	-0.03***	-0.02***	-0.04***	-0.02***
RedemptionNotice (30Days)	0.02***	-0.01***	0.04***	-0.03***
Lockup (months)	0.00**	0.01***	0.00	0.00***
PersonalCapitalDummy	-0.02***	-0.04***	-0.04***	-0.01***
HighWaterMarkDummy	-0.01***	-0.01***	-0.02***	0.01***
MgmtFee (%)	-0.01***	0.09***	0.05***	0.04***
IncentiveFee (%)	-0.01***	0.00	0.00	0.00***
Age (years)	-0.02***	-0.01***	-0.01***	-0.02***
ln (AUM)	0.01***	0.00**	0.00***	0.00
FlowPast1Y (%)	0.00**	0.00*	0.00	0.00***
AvgPast2YRet (% p.m.)	0.04***	-0.02***	-0.03***	0.01***
ln (MinInvestment+1)	0.00	0.01***	0.01***	0.01***
Leverage	0.06***	0.02***	0.04***	-0.01***
AdjR2 (%)	1.83	1.79	5.71	6.43

$* \ p \leq 0.05$, $** \ p \leq 0.01$, $*** \ p \leq 0.001$

$$Abnormal\ Performance_{i,t} = const + \gamma SuperiorGroupInd_{i,t} + \beta Control_{i,t-1} + \epsilon_{i,t}.$$

In this panel regression, γ captures the effectiveness of a machine learning model to differentiate the best funds from the worst ones. I expect better model to have higher value of γ. The result of this panel regression is presented in Panel C. The coefficient on SuperiorGroupInd of OLS is not statistically different from 0, which indicates no difference of risk-adjusted performance between the Superior group and the Inferior group after controlling fund characteristics. Although the coefficient of RF is marginally significant, it's economically small. The coefficients of GBRT and NN are statistically significant and economically large: after controlling for fund characteristics, the average difference of abnormal performance between the Superior group and the Inferior group are 0.21% and 0.29% per month for GBRT and NN, respectively. Such results are qualitatively similar to Table 2.3.

2.4.4 The Performance of Machine Learning Models with Macroeconomic Conditions

In the previous subsections, I showed that GBRT and NN are able to provide higher persistence than OLS model. They are promising to outperform OLS in terms of both raw returns and risk-adjusted returns, and the outperformance is robust under fund characteristics as control variables. In this subsection, I continue to explore the association of the outperformance with macroeconomic conditions, i.e., when does machine learning model tend to outperform?

I use a total of 6 macroeconomic indicators as the independent variables, which can be further grouped into 3 categories: economy state, market state, and uncertainty. The economy state has two variables: a recession indicator[5], which indicates when the business cycle steps into recession, and the percent change of unemployment rate. The market state variables include a bull-bear market indicator, which is calculated using the definition of bull/bear market provided in Lunde and Timmermann (2004), and value-weighted market return. The uncertainty variables include the percentage change of VIX index and the percentage change of US economic policy uncertainty[6]. All the 6 variables are of monthly frequency.

To examine the association between the outperformance of machine learning models and macros, I first calculate the difference of monthly portfolio-level returns between machine learning models and OLS, for both of the Superior group and the Inferior group, and obtain 6 time series: monthly return difference of the Superior portfolios between RF and OLS, GBRT and OLS, and NN and OLS, respectively, and 3 corresponding time series of the Inferior portfolios. Next, three additional diff-in-diff time series are obtained by calculating the difference of the difference of each machine learning model with OLS between the Superior group and the Inferior group. These nine monthly time series are then regressed on each of the macroeconomic variables. The results are shown in Table 2.5. Column 1 to column 3 show the results for the Superior group difference, column 4 to column 6 shows the results of the Inferior group difference, and column 7 to column 9 show the results of diff-in-diff.

[5] Data source: https://fred.stlouisfed.org/series/USREC
[6] Data Source: http://www.policyuncertainty.com/us_monthly.html

Table 2.5 Machine Learning Performances with Macroeconomic Conditions

Model	(1) Superior Diff: RF - OLS	(2) Superior Diff: GBRT - OLS	(3) Superior Diff: NN - OLS	(4) Inferior Diff: RF - OLS	(5) Inferior Diff: GBRT - OLS	(6) Inferior Diff: NN - OLS	(7) Diff in Diff: RF - OLS	(8) Diff in Diff: GBRT - OLS	(9) Diff in Diff: NN - OLS
Panel A: Business Cycle									
Recession Ind.	-0.03	0.49***	0.56*	-0.08	0.07	0.13	0.05	0.42	0.42
Constant	0.02	0.03	0.09	-0.02	-0.05	-0.11	0.03	0.07	0.20
AdjR2 (%)	-0.53	4.58	1.29	-0.49	-0.51	-0.51	-0.54	0.41	-0.29
Panel B: Unemployment Rate									
ΔUnemp	0.43	1.05***	1.55**	0.33	0.58	1.17	0.10	0.47	0.38
Constant	0.01	0.09	0.15	-0.03	-0.04	-0.10	0.04	0.13	0.25
AdjR2 (%)	0.40	5.12	2.89	-0.28	0.22	0.12	-0.54	-0.26	-0.50
Panel C: Bull vs. Bear Market									
Bull market ind.	-0.07	-0.40***	-0.62***	0.71***	0.63***	1.28***	-0.78***	-1.03***	-1.90***
Constant	0.06	0.39***	0.63***	-0.56***	-0.51***	-1.05***	0.62***	0.90***	1.68***
AdjR2 (%)	-0.39	4.93	3.09	7.79	5.36	4.76	5.41	8.52	7.89
Panel D: Value-Weighted Market Return									
Market returns	-0.02*	-0.04***	-0.05**	0.03	0.02	-0.03	-0.05**	-0.06***	-0.02
Constant	0.02	0.11**	0.18*	-0.04	-0.04	-0.08	0.06	0.15	0.26
AdjR2 (%)	1.19	6.39	1.88	0.69	0.08	-0.20	1.88	3.15	-0.47
Panel E: VIX									
ΔVIX	-0.01***	0.00	0.00	-0.01***	-0.01	-0.02**	0.00	0.01	0.02
Constant	0.03	0.09	0.16	-0.01	-0.03	-0.06	0.03	0.11	0.22
AdjR2 (%)	3.90	0.19	-0.52	3.44	0.45	1.72	-0.37	0.85	0.88
Panel F: Economic Policy Uncertainty									
ΔEPU	0.00	0.00	0.00	-0.01**	-0.01***	-0.03***	0.01	0.01***	0.03**
Constant	0.02	0.09	0.17	-0.01	-0.02	-0.05	0.03	0.11	0.21
AdjR2 (%)	0.18	-0.11	-0.37	2.77	3.94	5.18	0.36	3.19	2.77

$* \, p \leq 0.05$, $** \, p \leq 0.01$, $*** \, p \leq 0.001$

Panel A and Panel B show when the machine learning models outperform OLS under different economy state. For the differences in the Superior group, GBRT and NN performs significantly better than OLS when the economy turns into recession or when the unemployment rate increases. For the Inferior groups, there is no significant difference. And the diff-in-diff results are not significant either, though the coefficient estimates are positive.

Panel C and Panel D show when the machine learning models outperform OLS under different market state. When it is in bull market, GBRT and NN tend to do worse than OLS in the Superior group and do better than OLS in the Inferior group. Looking the other way around, when market is bearish, GBRT and NN tend to perform better than OLS in the Superior group and worse than OLS in the Inferior group. The diff-in-diff results in Panel C also confirms that when market is in bearish state, machine learning models are better at differentiating best funds from the worst ones than OLS. The result in Panel D are similar to Panel C except that for the Inferior groups the differences are not significant.

Panel E shows the performances under different levels of VIX. Both the Superior group and the Inferior group tend to do worse than OLS when VIX increases, there is no significant difference between the two groups.

Panel F shows the performances under different levels of economic policy uncertainty. While there is no difference for the Superior groups, the machine learning models of GBRT and NN tend to perform worse in the Inferior groups when economic policy uncertainty increases. And the ability of GBRT and NN to differentiate best funds from the worst ones when economic policy uncertainty is high is better than OLS, though it is not economically big (1% change in economic policy uncertainty is associated with 0.01%

change in the performance difference between the Superior group and the Inferior group of GBRT over OLS, and 0.03% change in NN over OLS).

In summary, Table 2.5 shows that GBRT and NN tend to perform better when the economy is in recession, when the unemployment rate increases, when the market is bearish, and when the uncertainty increases. The findings are consistent with the existing literature. For example, Cao, Goldie, Liang, and Petrasek (2016) demonstrate that hedge funds' superior performance is attributed to their ability to manage downside risk, and Sun, Wang, and Zheng (2018) show that hedge fund performance is persistent following weak markets but is not persistent following strong markets.

2.5 Summary

In this chapter, I use three machine learning methods, random forest, gradient boost random trees, and neural network, to predict future hedge fund performances. Using two-period portfolio sorting approach with 14 rolling window samples of 6-year, I show that portfolios with the best hedge funds constructed with machine learning methods are promising to provide higher performance persistence than OLS method. With the full sample portfolio analysis, I show that the Superior machine learning portfolios provide better performance than OLS in terms of both raw returns and Fung and Hsieh 7-factor adjusted returns, while the Inferior machine learning portfolios do not have significant performance difference from OLS. In a panel regression with hedge fund characteristics, I show that the outperformance of the Superior group of machine learning methods are not driven by fund characteristics. And in the analysis on the association of outperformances of machine learning methods with macroeconomic conditions, I show that machine learning methods tend to perform better than OLS when the economy is in recess, when the market is bearish, and when the level of uncertainty increases.

Chapter 3

An Application of Reinforcement Learning on Algorithmic Trading

3.1 Background

Machine learning research had many significant advances in recent years, transforming our technology. In many industries, machine learning has found valuable applications. For example, in financial services industry, machine learning has been applied to provide automated financial guidance and services (robo-advisors), fraud detection, loan and insurance underwriting, risk management, etc. As an important area in machine learning, reinforcement learning (RL) is increasingly getting attention in the area of algorithmic trading.

In a typical setting of algorithmic trading, a computer program called director manages the execution. By construction, the director executes its orders indirectly by outsourcing the orders to its two sub-agents: a passive order agent and an aggressive order agent[7] (aggressive agent hereafter). Normally orders are mainly executed by the passive order agent to save execution cost, aggressive agent only comes into play in certain cases, such as when execution is in urgency. The advantage of executing orders aggressively is that orders are almost always filled immediately. However, the downside is that it would incur an extra bid-ask spread to the transaction cost. In this chapter, transaction cost is defined as the differences between the stock price when a trading decision is made and the average price of the stock when the transaction is actually executed, or implementation shortfall.

However, executing a big chunk of aggressive orders at once is very costly, as it would trigger huge market impact. A natural solution is to smooth out the aggressive order

[7] An order is passive when traders set a price that stock must reach before they go ahead with buying or selling. In contrast, aggressive orders are when a trader executes the order to buy or sell straightaway. Passive orders are also called limit orders, aggressive orders are also called market orders.

execution: instead of executing a single big chunk of quantity at once, it gradually executes a series of smaller orders across time, thereby imposing a smaller market impact and potentially reducing the transaction cost. A natural way to do this is to follow a linear schedule, which is to execute evenly through time.

Although executing aggressive orders with a linear schedule within a time range is better than doing so at a single point in time, it is definitely not optimal, since this simple strategy does not take any book information[8] or trade signals[9] into account at all. Ideally, we should be able to form an opinion about the market based on the book information and form trade signals such that we would behave differently under various market conditions. Therefore, the distribution of the orders executed would be hardly the same as uniform.

To take the advantage of book information, there are multiple methods: model-based methods and model-free methods. Model-based methods assume that the market evolves according to some pre-imposed statistical processes. Such methods can often run into a dilemma: on one hand, some models are too simplified to describe the market dynamic; on the other hand, complex models often face the curse of dimensionality, i.e., data needed to calibrate the model increases exponentially with the number of model parameters. In either case, the models are of little practical use.

On the contrary, model-free methods, as the name indicates, do not assume a model for the market. One of such methods is RL, which is concerned with an agent that learns an optimal way to accumulate its rewards. RL consists of several key elements: an environment, an agent, and a reward function. Simply speaking, we can interpret the order execution process in the real world into these elements of RL:

[8] Book refers to limit order book.
[9] A trade signal is a trigger for action, either to buy or sell a security or other asset, generated by analysis.

- A trading environment (the market) that takes orders and gives responses.
- An agent that executes orders according to different market conditions.
- A series of rewards, such as market return or price movements, that are associated with the market condition and the agent's action.

This abstraction of a real-world trading problem fits well into an RL framework: the market evolves constantly and gives the agent necessary information, the agent decides whether to trade or not, and if so, the quantity to execute, and reduces transaction cost as much as possible.

Under the framework of RL, a model that characterizes the transition probability of the environment is not needed. Instead, RL focuses on data, and the data speaks for itself as the training process goes. Using a market simulator, we generate and feed real market data to the agent in terms of episodes[10] and let the agent explore and improve. The agent must be able to execute the required quantity within the given time and minimize the transaction cost. Typically, the agent must go through a large number of episodes before it stabilizes and converges.

My data is the daily data of trade and quote for each stock from all major Asia Pacific markets from opening to close. Trade data contain all trades for a stock, and each trade record includes the timestamp at which the trade took place, stock price, and trade size. Quote data contains the time series of book information, including timestamp, best bid and ask price, and the available quantity.

To train the agent, I use deep Q-network (DQN) with temporal difference of 20 steps, for which implementation details will be discussed in Section 3.4. Once the agent is trained,

[10] An episode is defined as a sub-order, which has a required quantity and a time horizon to execute.

I employ arrival price slippage[11] to measure the performance of the agent. Furthermore, I benchmark the performance of the agent against an agent with linear execution schedule, which I call a linear agent hereafter. I train and test the agent in all major stock markets in Asia. In each market, I test the agent on the 100 most actively traded stocks. For each stock, I randomly generated 100 episodes to evaluate the performance. The results show that the aggressive agent outperforms the linear agent in major Asia Pacific markets: on average the aggressive agent outperforms the linear agent by 0.11 bps to 1.12 bps in terms of arrival slippage for in-sample data, and 0. 12 bps to 0.69 bps for out-of-sample data.

I notice that there are several previous works that have examined the prosperity of RL in trading execution. In particular, Almgren and Chriss (2001) provide an abstract mathematical framework for trade execution problems. Nevmyvaka, Feng, and Kearns (2006) explore a large-scale empirical application of RL on microstructural data. Hendricks and Wilcox (2014) extend Almgren and Chriss (2001) with a linear market impact model and apply RL in South Africa equity market. Ritter (2017) explores the application of RL under risk-aversion settings. Ritter and Tran (2018) provide an RL framework for trading problems with continuous states. And Capponi and Cont (2019) examine the empirical relationship between market impact and volatility. To the best of my knowledge, this project is the first to apply RL with multi-step temporal difference DQN method in trading execution in Asia Pacific stock markets.

The chapter proceeds as follows. In Section 3.2, I give a brief introduction of RL. In Section 3.3, I present the trading problem formally as an RL formulation. In Section 3.4, I

[11] Arrival price slippage is the difference between realized execution cost and hypothetical cost at the price when order arrives.

discuss the implementation details. In Section 3.5, I present agent's behavior and performance analytics. And in Section 3.6 I conclude.

3.2 Overview of Reinforcement Learning

As an important area of machine learning, RL is concerned with an agent who learns an optimal way to maximize cumulative rewards in a certain environment. Specifically, the basic ingredient of an RL problem is a Markov decision process (MDP), which is a discrete time stochastic control process that consists of three basic elements: environment, agent, and reward. Typically, the MDP evolves as follows:

1) The environment gives out a set of states M.
2) Observing M, the agent performs an action A according to its policy $\pi = \pi(M)$.
3) The environment transits from state M to state $\widetilde{M}$ under the agent's action A, according to its transition probability P.
4) The agent receives a reward R as a function of M, $\widetilde{M}$, and A.
5) The environment gives out the next state $\widetilde{M}$, and the process continues as such.

The exact solution of an MDP can be derived with dynamic programming techniques when the dynamics of the MDP (i.e., the transition probability P) is completely known. When it is unknown or when the MDP is so large that a dynamic programming solution becomes infeasible (due to the curse of dimensionality), RL comes into play.

Unlike supervised learning, the data of RL problems does not have labels. Therefore, an RL agent would not be dictated to perform the right action. Instead it is on its own. In fact, an RL agent often behaves randomly initially and acts sub-optimally early in the training process. Such random behavior is what we call exploration. As it learns from its experience and gradually improves itself (by updating its policy π) as the training goes, the agent tries to perform the same actions that are associated with higher rewards and to avoid bad actions associated with lower rewards. Hence the term *reinforce*.

3.3 The Aggressive Trading Problem: A Reinforcement Learning Formulation

3.3.1 Cost Minimization under No information

As described in Section 3.1, what we face is the problem to execute certain number of shares within a certain time horizon. We adopt the trading model used in Almgren and Chriss (2001). Suppose we have in total a quantity of X shares to execute within a time horizon T, which we refer to as one episode. We view this time horizon T as a series of N discrete intervals with an arbitrary small duration of $\tau = \frac{T}{N}$. Orders can be placed in the stock market at any time. We denote t_k as the end time of the kth interval and S_k as the stock price at t_k. In each interval i, we place n_i shares. After k intervals, we have executed $\sum_{i=0}^{k} n_i$ shares, with the remaining quantity being $X_k = X - \sum_{i=0}^{k} n_i$, i.e., $n_k = X_{k-1} - X_k$. We require that $X = \sum_{i=0}^{N} n_i$, i.e., order is fully executed when time is up. We summarize our notations in Table 3.1.

Table 3.1 Notations of A Trading Model

Notation	Description
S_k	Fundamental stock price at interval k
$\widetilde{S_k}$	Observed instantaneous execution price
X	Total quantity to execute
N	Total number of intervals
T	Terminal time of the order
t_k	Time point at the beginning of the kth interval, with $t_0 = 0$ and $t_N = T$.
τ	Length of each interval
X_k	Remaining quantity to execute at the kth interval, with $X_0 = X$ and $X_N = 0$.
n_k	Quantity executed at the kth interval, $n_k \geq 0$.
$v = n_k/\tau$	Speed of trading

Assume a sell order is being executed. Further assume no interactive effects among stocks, i.e., the prices of any pair of stocks is uncorrelated, the stock price S_k follows a random walk with a drift term caused by stock sale

$$S_k = S_{k-1} + \sigma\tau^{1/2}\xi_k - \tau g\left(\frac{n_k}{\tau}\right), \tag{1}$$

where

$$g(v) = \gamma v \tag{2}$$

is the linear permanent price impact with respect to the trading speed v with impact coefficient γ, $\xi_k \sim N(0,1)$ is an i.i.d. normal random variable with zero mean and unit variance, and σ is the volatility of the stock. We regard S_k as the equilibrium price, where executions happened in the interval k change it as well as bring it to a new equilibrium. The actual stock price $\widetilde{S_k}$ for each execution in the kth interval depends on the previous permanent price S_{k-1} and the executions happened in the market

$$\widetilde{S_k} = S_{k-1} - h\left(\frac{n_k}{\tau}\right), \tag{3}$$

where

$$h(\upsilon) = \eta\upsilon + \frac{c}{2} \tag{4}$$

is the linear temporary price impact with respect to the trading speed υ with η being the coefficient of the impact and c being the price difference between the best buy and sell prices. Intuitively, the price movement from S_{k-1} to $\widetilde{S_k}$ is short-term, since the impact by the execution of n_k shares is instantaneous. In other words, although the order execution may break the balance of the demand and supply, such balance would resume shortly.

However, its temporary impact may be carried over. That is, S_k has a tendency to revert to the original level of S_{k-1}, but not fully, as illustrated in Figure 3.1. We call such phenomena permanent impact, as the effect remains in the subsequent market dynamics.

In each interval k, which lasts for τ seconds, the agent decides a portion n_k of the order to execute. Our target is to complete the order with an optimal trajectory $n^* = (n_0, n_1, n_2, \dots, n_N)$. The objective is to minimize arrival slippage, defined as

$$U(n) = XS_0 - \sum_{k=0}^{N} \widetilde{S_k}\, n_k, \tag{5}$$

which is the difference between the total cash collected by execution and the hypothetical amount if the order is executed at the price when the order is received.

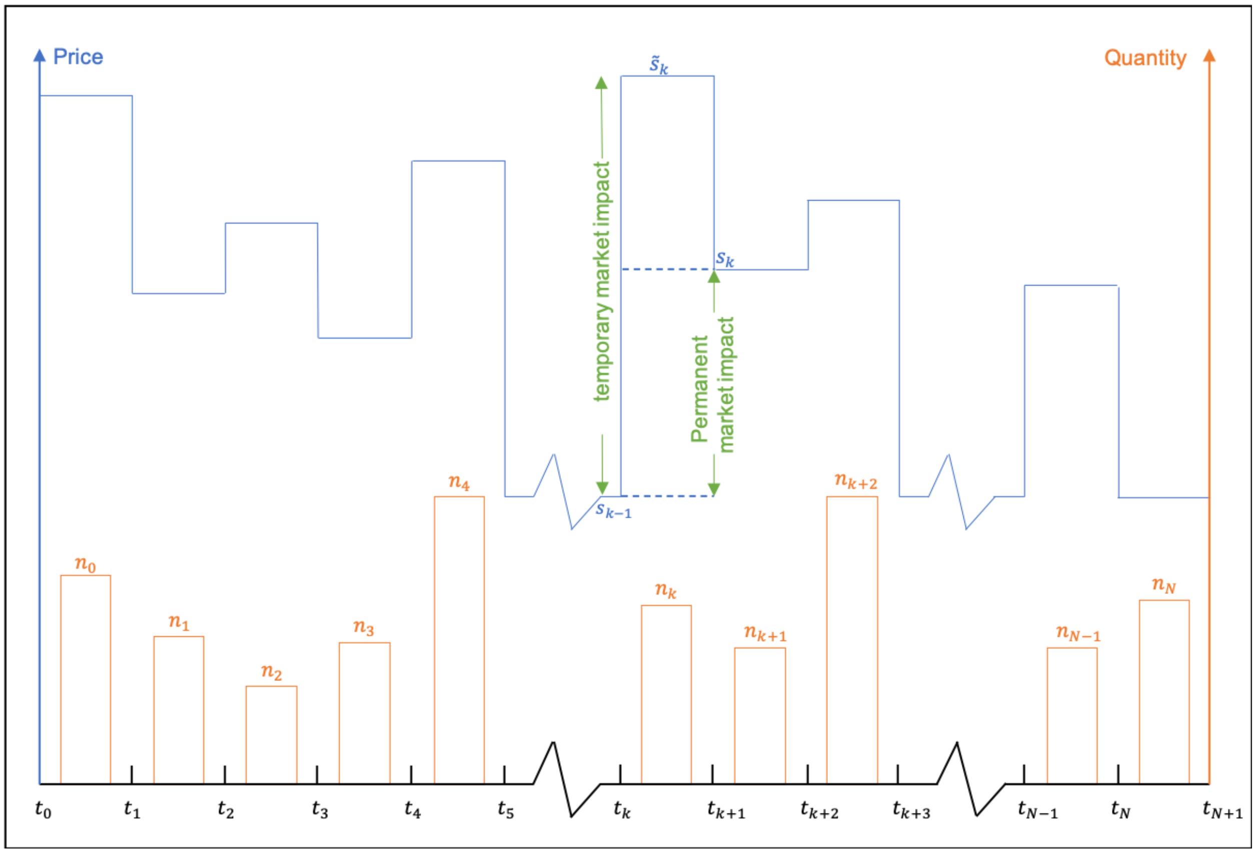

Figure 3.1 An Illustration of Market Dynamics

Let us first solve the optimal trajectory if the agent had no book information or trade signals. According to (1) and (3), and the fact that $n_k = X_{k-1} - X_k$, (5) can be rewritten as

$$U(n) = \sum_{k=0}^{N} \left[\tau g\left(\frac{n_k}{\tau}\right) - \sigma\tau^{1/2}\xi_k\right] X_k + \sum_{k=0}^{N} n_k h(\frac{n_k}{\tau}), \tag{6}$$

for which the expectation and variance are then

$$E[U(n)] = \sum_{k=0}^{N} \tau X_k g\left(\frac{n_k}{\tau}\right) + \sum_{k=0}^{N} n_k h(\frac{n_k}{\tau}) \tag{7}$$

and

$$Var[U(n)] = \sigma^2 \sum_{k=0}^{N} \tau X_k^2. \tag{8}$$

Furthermore, if we substitute (2) and (4) into (7), we can further expand (7) as

$$E[U(n)] = \frac{1}{2}\gamma X^2 + \frac{c}{2}X + \frac{\eta - \frac{1}{2}\gamma\tau}{\tau}\sum_{k=0}^{N} X_k^2. \tag{9}$$

There are three key observations from (9):

- The term $\frac{1}{2}\gamma X^2$ is due to permanent impact, as γ inherits from (2).
- The term $\frac{c}{2}X$ is due to temporary market impact and is linear in c, i.e., how large is the gap between the best buy and sell price.
- Most importantly, the last term is quadratic to each slice of the execution schedule. Therefore, the optimal schedule to reach minimum expected arrival slippage is

  $$n^* = \frac{X}{N} \cdot 1, \tag{10}$$

  which means $n_0^* = n_1^* = \cdots = n_N^* = \frac{X}{N}$, i.e., a linear time schedule.

Interestingly, the result in (10) implies that the minimum payment that we expected to make is $\frac{X}{N}\sum_{k=0}^{N}\widetilde{S_k}$ by following a linear schedule.

3.3.2 Utilizing Book information and Trade Signals

Although the optimal trading schedule has been mathematically derived to be a linear schedule, we can use additional information in the market to seize the opportunistic benefits to further lower the execution cost. From an economic point of view, deviating from the otherwise optimal linear schedule (10) would incur extra cost, but utilizing trade signals might achieve extra benefit. Next we will consider the tradeoff between extra benefit and extra cost.

Suppose that the return between t_k and t_{k+m} is α_k, where m is the prediction horizon, and we have no predictions for other future horizons. In order to achieve the benefit, the agent intentionally trades ϵ_k more shares in the current time interval. The extra gain from utilizing the signals is hence

$$\epsilon_k \alpha_k \tau. \tag{11}$$

Note that ϵ_k can be either positive or negative, i.e., the agent can execute faster than a linear schedule or slower. Also note that even if α_k is large, ϵ_k in (11) would not be X_k, the number of all the remaining quantities: the agent would not be myopic to execute all the remaining quantities only based on one promising signal, since there may be a chance the it could get an even better signal in the next interval.

Once the additional ϵ_k is executed, the agent needs to resume the linear schedule thereafter. Since the total quantity to execute is fixed, at each of the next $N-k$ time intervals the agent would execute $\frac{\epsilon_k}{N-k}$ shares less in order to offset the extra ϵ_k shares executed at the current interval k. That is, the agent executes orders with the following schedule

$$\hat{n}_i(\epsilon_i) = \begin{cases} \dfrac{X}{N}, \quad i = 0, \dots, k-1 \\ \dfrac{X}{N} + \epsilon_i, \quad i = k \\ \dfrac{X}{N} - \dfrac{\epsilon_k}{N-k}, \quad i = k+1, \dots, N \end{cases} . \tag{12}$$

Given the additional shares ϵ_i executed in time interval t_k, we derive the expected extra cost as

$$E[U(n^*) - U(\hat{n})] = \frac{\eta - \frac{1}{2}\gamma\tau}{\tau}\left[\epsilon_k^2 - \frac{\epsilon_k^2}{N-k}\right], \tag{13}$$

which is the difference of the expected arrival slippage between the original linear schedule (10) and the deviating schedule (12).

By leveraging (11) and (13), the total cost difference by utilizing the signal is

$$F(\epsilon_k) = -\epsilon_k \alpha_k \tau + \frac{\eta - \frac{1}{2}\gamma\tau}{\tau}\left[\epsilon_k^2 - \frac{\epsilon_k^2}{N-k}\right], \tag{14}$$

where a negative cost difference means that overall, we gain from deviating from the linear schedule. We note that, in each time interval t_k, we might have some signal about the future $\alpha_t, t = k, k+1, \dots, N$. However, each α_k is not known in advance. We decide ϵ_k in each time interval by modeling the decision as a stochastic process with minimizing (14) across the trade duration

$$\min \sum_{k=0}^{N} E\left(\beta^k F(\epsilon_k)\right), \tag{15}$$

where $\beta \in (0,1)$ is the discount factor. Let $I(\epsilon_k)$ be the minimum total cost incurred if we decide to trade ϵ_k more shares than the linear schedule, the execution of $\frac{X}{N} + \epsilon_k$ shares not only transits the status of the remaining quantity from X_k to X_{k-1}, but also brings us a cost of $F(\epsilon_k)$. The Bellman equation (Bellman, 1953) in (15) can be written as

$$I(\epsilon_0) = \min_{\epsilon_0}\left\{F(\epsilon_0) + \beta\left[\min_{\{\epsilon_k\}_{k=1}^{N}} \beta^{k-1} F(\epsilon_k)\right]\right\}. \tag{16}$$

Generally (16) can be solved by dynamic programming if the dynamics of α_k is known. However, in practice we seldom do. Therefore, an exact optimal solution is very hard to obtain. Fortunately, under the framework of RL, we can train a neural network to represent the unknown dynamics and help us make decisions. In the next section we will present how we implement the RL algorithms and approximate optimality.

3.4 Implementation

As discussed in Section 3.2, there are three elements in RL: environment, agent, and rewards. Next I will present the details one by one.

3.4.1 Trading Environment

I use a market simulator to set up the environment to generate trading episodes. The simulator takes market data (quote and trade) and trade signals as input, and generates episodes given date and time horizon. In each time interval of an episode, the simulator sends out a state vector that consists of two parts: book-information-related variables and trade signals. This vector is fed to the agent as input. Note that in practice, the signals are very noisy: for intraday trade signals, the R-squared of signals on returns is often less than 1%. To assess the agent, I let the simulator to generate both perfect and more realistic signals. I first train the agent with 1-min forward-looking return, which is a 100% accurate signal, and then gradually add noise to the signal to test the stability of the agent. Training in this way provides an extra benefit: it modularizes the development process, such that the agent research and signal research can be performed independently.

3.4.2 Agent

I use deep Q-network (DQN) to train the agent. There are challenges due to the following reasons. First of all, the agent evaluates the environment states and acts accordingly every three seconds, the signals it receives are on one-minute frequency. Therefore, the signal horizon and the action horizon are different. Secondly, for many APAC markets, stock prices do not change during a period as short as three seconds. Thus, the series of the three-second return is very sparse. This would slow down the convergence of training, since most of the time, the rewards would be zero. To solve these problems, we use DQN with temporal difference of 20 steps, which accumulates rewards for 19 steps to iterate the action-value function, for which details will be discussed in Section 3.4.5. By doing so, it forces the action horizon to be in line with the signal horizon. The target becomes less sparse as well.

3.4.3 Actions

Ideally, the action space of the agent includes all possible number of shares that can be placed. However, there can be potentially arbitrarily large number of shares to execute, causing the Q-network too long to converge. Thus, mapping actions to the number of shares to execute is incompatible with DQN. Furthermore, different stocks have different characteristics and number of shares outstanding. Directly mapping actions to the number of shares to execute would lose generality that the agent has to be trained on a per stock basis instead of a group of stocks. Training on a per stock basis is sometimes undesired: since some stocks are inactively traded, the lack of data of such stocks would make it hard to train a meaningful network.

My solution is to map the actions to a 5-dimensional placement vector instead of the number of shares. Specifically, at each step k, we first compute the quantity that would be executed by a linear agent, denoted by q_k. Note that q_k is dynamically changing based on the number of shares that have been executed in the past. For example, suppose the agent receives an order to execute 10000 shares within 10 minutes. Since the agent evaluates and responds to the environment every 3 seconds, it has a total of 200 steps. Thus, initially $q_0 =$

$\frac{10000}{200} = 50$. If, for some reason, the aggressive agent deviates the linear schedule, it has executed 6000 shares at step 100. Then for the next step, $q_{101} = \frac{10000-6000}{100} = 40$. In other words, q_k is the linear quantity recalculated at each step. The placement vector A_k is the following, which is different multiples of q_k from 0 to 2:

$$A_k = [0\ 0.5\ 1\ 1.5\ 2]^T * q_k.$$

At each step, the agent could choose one of the five actions. In this way, the agent would be generic to different stocks, since the decision is to choose which multiple of q_k to execute, not the actual quantity.

3.4.4 Rewards

As discussed previously, deviating from a linear schedule due to signal utilization has two effects: the penalty from deviating and the benefit from alpha harvesting. According to (11) and (13), the positive effect is linear in ϵ_k and the negative is quadratic in ϵ_k. In practice, we design the positive reward as the product of step return r_k and ϵ_k

$$\epsilon_k r_k \tag{17}$$

where ϵ_k is defined as

$$\epsilon_k = \frac{n_k - q_k^0}{X} \tag{18}$$

Here, (18) is the percentage difference of execution quantities between the aggressive agent and the linear agent, normalized by the total quantity X. Note that we choose not to use the signal α_k in (17) for two reasons. The first reason is that though in some of our earlier experiments, using (11) as the positive reward function works under perfect α_k, α_k is very noisy in reality. This means the rewards being collected is noisy as well if we use α_k in reward function. Accumulating noisy rewards makes convergence harder. The second reason is that using signals directly in the reward function would lose the flavor of RL: the agent is dictated to perform a "right" action that is in line with the signals.

Next, we design our negative effect reward as

$$\varphi \epsilon_k^2 \sigma^2 \tau. \tag{19}$$

In (19), σ is the annualized volatility of the stock. It is shown in Capponi and Cont (2019) that σ^2 is positively related to market impact, so we use σ^2 as the proxy for market impact. We define φ as the coefficient of risk-aversion, to control the extent of deviation: the larger the φ, the more negative the reward, and the smaller the deviation in equilibrium. When φ is very large, the aggressive agent would follow a linear schedule. In practice, this φ is a hyper-parameter to be tuned.

3.4.5 Algorithm

As discussed earlier, we use DQN with temporal difference of 20 steps to train the agent. In general, we face two major problems with DQN. The first one is that our training samples are not i.i.d., as they are generated sequentially from an episode. To solve this problem, we use an experience replay, which is a collection of tuples of current state, action, reward, and next state. At each step, instead of training the agent with the current transition sample, we add this sample data to the experience replay, and randomly sample a batch of data from the replay buffer and train agent on this batch. In this way, samples in the batch are less correlated. The second problem is that the target of the Q-network is constantly changing. In particular, we are trying to learn a Q-value function with the following updating scheme

$$Q_{k+1} \leftarrow (1-\alpha_k)Q_k(s_k, a_k) + \alpha_k \left[r_k + \gamma \max_{a_k'} Q_t(s_{k+1}, a') \right].$$

By construction, Q value is changed each time update the network. However, the target actions are generated with the same Q-network. The labels are constantly changing as well. As a result, training could be unstable. To overcome this problem, we use a target network. A target network generates actions, provides the associated predicted Q-values, and has the same architecture of the primary network. The primary network is updated at each step, while the target network is updated periodically by copying parameter values of the primary network. In this way, the target is not constantly moving. The target network improves the stability and provides faster convergence.

Besides the above two problems, we also face the problem of sparse return and unmatched signal horizon, as we discussed earlier. The solution is to use multi-step temporal difference instead of one-step temporal difference in standard DQN training. In Figure 3.2, I show the pseudo code of the algorithm for training.

Finally, we choose Huber-loss as the loss function to alleviate the impact of outliers

$$L_\delta(\zeta) = \begin{cases} \frac{1}{2}\zeta^2, & \text{if } |\zeta| < \delta \\ \delta\left(|\zeta| - \frac{1}{2}\delta\right), & \text{otherwise} \end{cases}.$$

```
Initialize replay memory $D$ with capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $Q'$ with weights $\theta' = \theta$
**For each** episode = 1 to $M$ **do**
    Get initial state $S_0$
    **While** episode not done **do**
        **If** $p \sim U[0,1] \leq$exploration rate **then** select a random action $a_t$
        **Else** $a_t = argmax_a Q(s_t, a; \theta)$
        **End if**
        Execute $a_t$ and obtain $r_t$ and $s_{t+1}$
        Store the transition $(s_t, a_t, s_{t+1}, r_t)$ in $D$
        **If** size of $D \leq$ mini-batch size **then**
            Sample a mini-batch $B$ from $D$
            **For each** $b_i = (s_i, a_i, s_{i+1}, r_i)$ in $B$ **do**
            Compute target value
                $y_i =$
$\begin{cases} r_i, & \text{if episode done at } s_{i+1} \\ \sum_{k=i}^{i+19} \gamma^{k-i} r_k + \gamma^{20} \max_{a'} Q'(s_{i+20}, a'; \theta'), & \text{otherwise} \end{cases}$
            **End for**
            Update $\theta$ by performing gradient descent on $Q$ with mean-
```

Figure 3.2 Deep Q-Learning Algorithm with Temporal Difference of 20 Steps

3.5 Experiments

In this section, we present the performance analytics of the agent.

3.5.1 Pre-cautionary Executions and Dilatory Executions

After the aggressive agent is trained, we let it perform on random orders to examine its behavior. We observe that the agent demonstrates precautionary execution and dilatory execution behavior when appropriate. Pre-cautionary executions refer to a series of execution intervals in which the agent executes shares at a higher speed (higher number of shares) than the linear agent. On the contrary, dilatory executions refer to those intervals in which the agent executes shares at a lower speed (lower number of shares) than the linear agent. Typically, the aggressive agent shows pre-cautionary execution behavior when the stock price is moving against it. That is, when the stock price increases for a buy order or decreases for a sell order, it executes at a higher speed to avoid higher cost later. On the other hand, it shows dilatory execution behavior when the stock price is moving for it. That is, when stock price decreases for a buy order or increases for a sell order, it executes at a lower speed to take advantage of a better price later on.

3.5.2 Typical Execution Schedule

In Figure 3.3, I show the comparison of a typical execution schedule for a sell order on a high-tech stock listed in Hong Kong Stock Exchange between the aggressive agent and the linear agent. The order lasts for 10 minutes. Figure 3.3a shows the stock price. Figure 3.3b shows the execution progress of both the aggressive agent and the linear agent. Note that the execution progress of the linear agent is always the diagonal straight line from lower left to upper right. For the actual execution schedule, the slope of the curve shows the execution speed. We observe that when the stock price is decreasing (from 15:27 to 15:29 in Figure 3.3a) the aggressive agent executes at a higher speed (from 15:27 to 15:29 in Figure 3.3b) to avoid selling at a lower price later. When the stock price is increasing (from 15:30 to 15:32 in Figure 3.3a), the aggressive agent executes at a lower speed (from 15:30 to 15:32 in Figure 3.3b), selling at a higher price later. Figure 3.3c shows the weighted average execution price for both agents: the aggressive agent clearly outperforms the linear agent by selling shares at a higher average price.

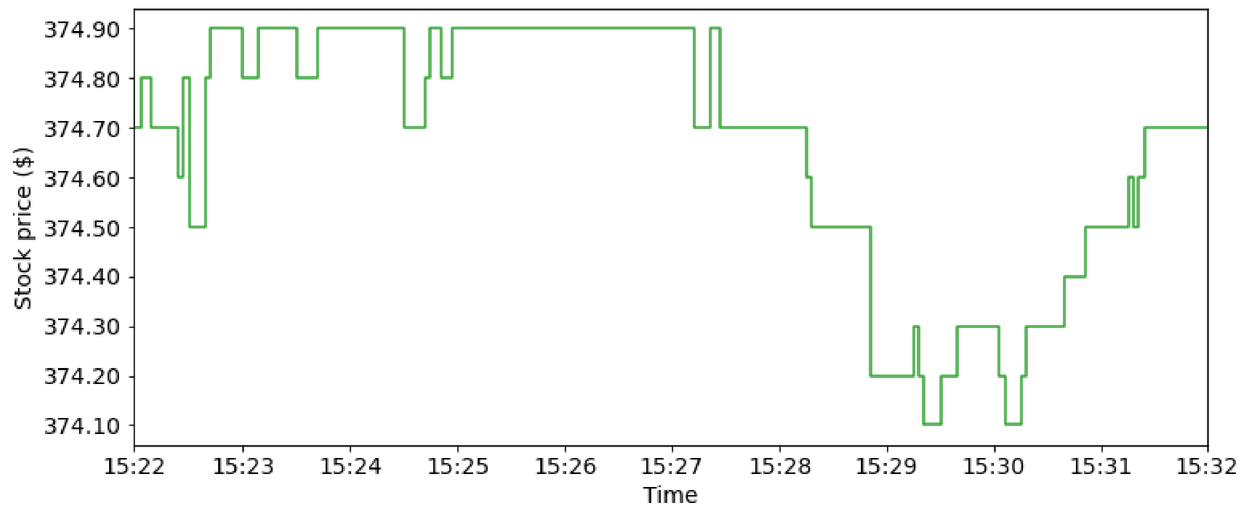

Figure 3.3a A Sample Order - Stock Price

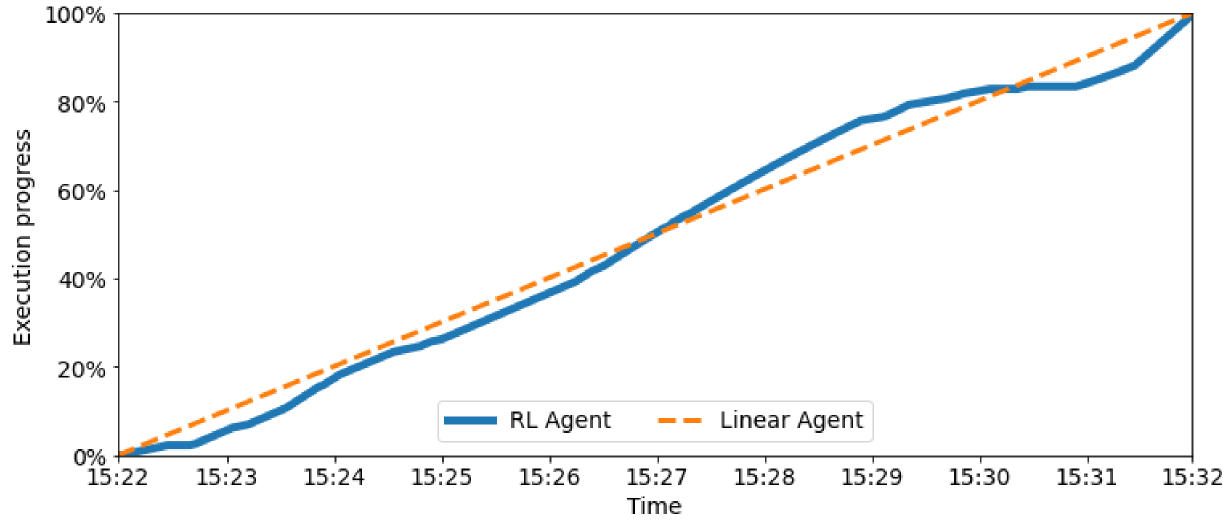

Figure 3.3b A Sample Order - Execution Progress

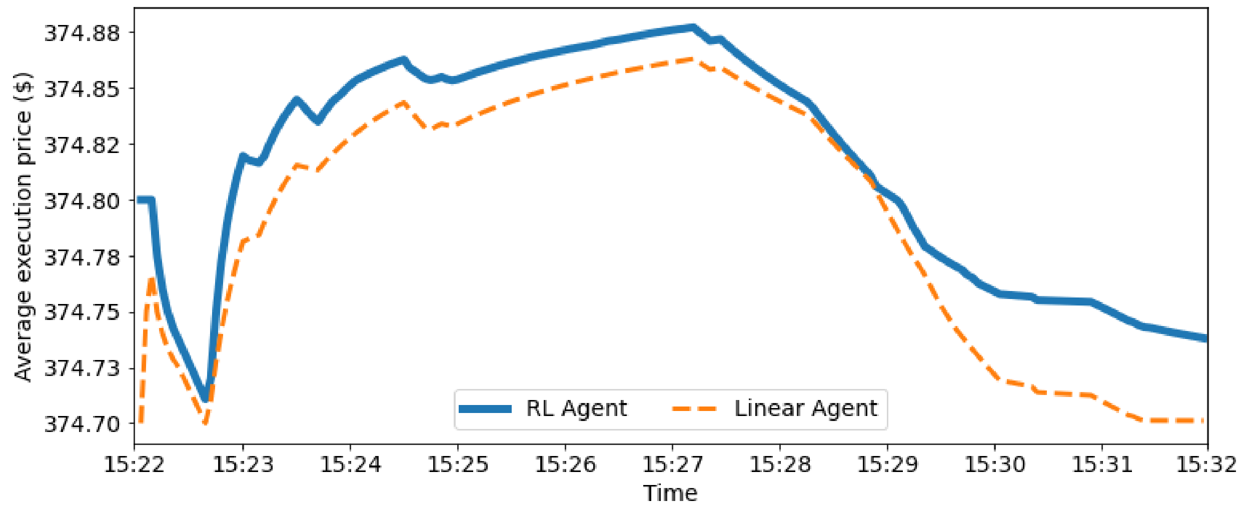

Figure 3.3c A Sample Order - Average Execution Price

3.5.3 Network Visualization

In addition, to better understand how the Q-network performs under different signal values, we plot the output values of the Q-network in Figure 3.4 for different values of trade signals. For each input signal value, the network outputs 5 Q-values, corresponding to each of the 5 actions, from which the agent chooses the one with the highest Q-value to perform. In Figure 3.4, the x-axis represents signal values from negative to positive and y-axis represents each of the 5 actions. Actions with lower Q-values are in brighter colors whereas those ones with higher Q-values are in darker colors. The best action is highlighted in navy blue. Figure 3.4 shows that as the signal value increases, the best action increases monotonically as well.

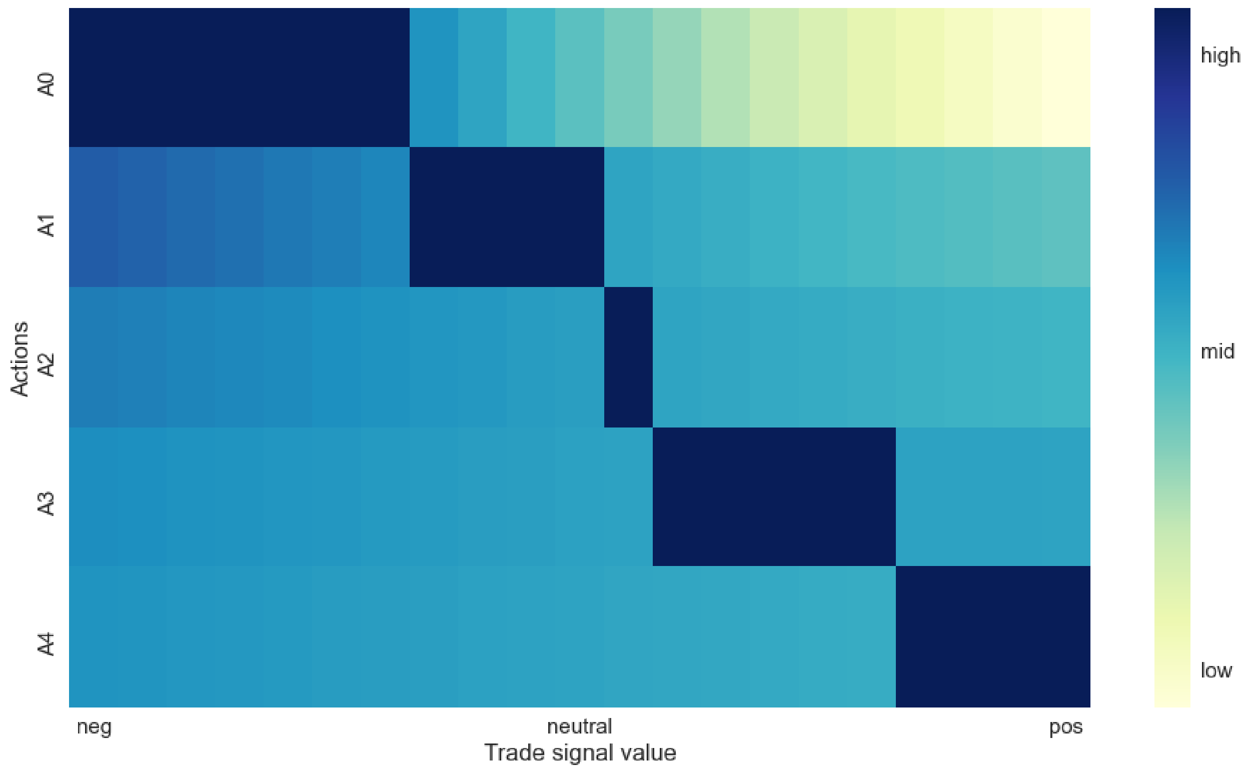

Figure 3.4 Q-Network Visualization

3.5.4 Performance Analytics

To understand how the agent performs in different markets, we test the performance of the agent in each of the 11 Asia Pacific stock markets in Table 3.2. For each market, we select 100 most actively traded stocks and train the agent on each stock with data from May 2019. After training, for each stock, we let the trained agent perform on 100 randomly generated episodes and benchmark the average performance against the linear agent. Then we calculate the average outperformance of the aggressive agent across all the stocks in each market in the second column and show the corresponding t-statistics in the third column. We also test the out-of-sample performance of the aggressive agent with data from June 2019, the results are shown in column 4 and 5, respectively. Table 3.2 shows that except for Singapore and Thailand, for in-sample data, the agent consistently outperforms the simple linear agent by 0.11 to 1.12 basis points (bps), which are statistically significant. For out-of-sample data, the agent performs similarly, it outperforms the linear agent by 0.12 to 0.69 bps.

Table 3.2 Aggressive Agent Performance by Asia Pacific Stock Markets

Market	In-sample Slippage Advantage (bps)	Out-of-sample Slippage Advantage (bps)
Australia	0.11**[a]	0.12***
China	1.12***	0.69***
Hong Kong	0.32***	0.25***
India	0.69***	0.60***
Indonesia	0.20**	0.18**
Japan	0.37**	0.21
Korea	0.36***	0.49***
Malaysia	0.13*	0.20***
Singapore	0.00	0.02
Taiwan	0.35***	0.28***
Thailand	0.10	0.02

[a] $* \, p \leq 0.05$, $** \, p \leq 0.01$, $*** \, p \leq 0.001$

Finally, as we discussed in the previous sections, in this project I am using forward-looking one-minute return as the trade signal, in order to separate the negative effect of inaccurate and noisy signals. Now that the agent works as desired under perfect signal, I move on to test its validity when the signals are noisy. To test the agent under such scenarios, I introduce the definition of signal-to-noise ratio (SNR). Assume in practice we observe a trade signal $\alpha_k^O = \alpha_k^T + \mu_k$, where α_k^O is the observed noisy signal, which is the sum of the unobserved true signal α_k^T and a noise μ_k. SNR is defined as the ratio of the variance of α_k^T over the variance of μ_k, that is,

$$SNR = \frac{\sigma^2(\alpha_k^T)}{\sigma^2(\mu_k)}.$$

I select the same stock as in Figure 3.3 to demonstrate the effect of SNR on the performance of the aggressive agent. The agent is trained under a range of SNR's from 1% to infinity, i.e., pure signal. For each SNR, as before, the agent is run on 100 randomly generated episodes. I plot the outperformance of the aggressive agent over the linear agent against SNR. The results are shown in Figure 3.5. As expected, as the signal becomes noisier, it is harder for the agent to learn to utilize the signal, and the performance deteriorates. However, even under 1% SNR, the aggressive agent still outperforms the linear agent by about 0.5 bps per order. Therefore, the aggressive agent is promising to outperform the simple linear agent even under very noisy environments.

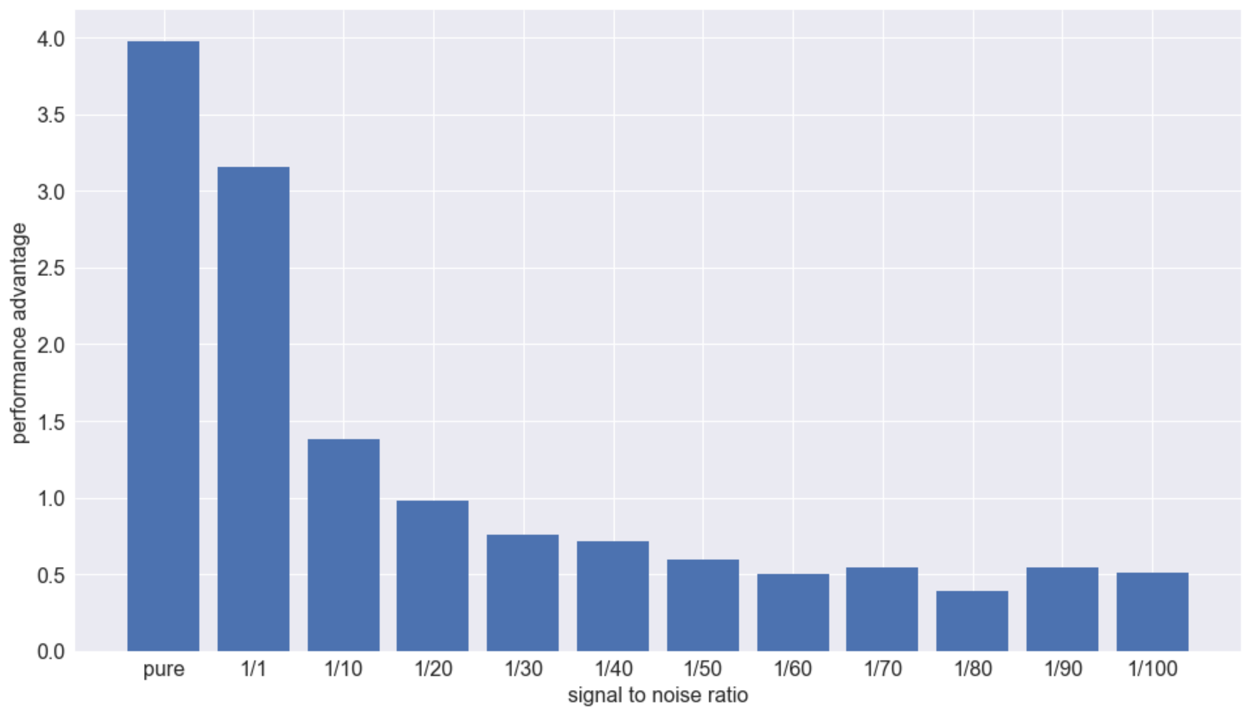

Figure 3.5 RL Agent Performance Advantage under Various SNRs

3.6 Summary

Reinforcement learning applications are increasingly being explored in Finance industry. The problem of stock trading provides an ideal setting for reinforcement learning. In this chapter, using reinforcement learning with deep Q-network, we successfully trained an agent that executes orders aggressively. A well-trained agent exhibits the behaviors of pre-cautionary execution and dilatory execution when appropriate. Our back-test analysis in APAC stock markets shows that the aggressive agent outperforms an agent with linear execution schedule by 0.11 bps to 1.12 bps on average per order for in-sample data, and 0.12 bps to 0.69 bps for out-of-sample data. Furthermore, as expected, the outperformance of the aggressive agent over the linear agent increases with increasing SNR.

In this chapter, the actions of the agent are encoded into a finite discrete placement vector and trained with DQN method. For future research, we can use the number of shares directly as the raw action and train the agent with policy-based methods.

REFERENCES

Ackermann, C., McEnally, R., & Ravenscraft, D. (1999). The Performance of Hedge Funds: Risk, Return, and Incentives. *Journal of Finance, 54*, 833-874.

Agarwal, V., & Naik, N. Y. (2000). Multi-Period Performance Persistence Analysis of Hedge Funds. *The Journal of Financial and Quantitative Analysis, 35*(3), 327-342.

Almgren, R., & Chriss, N. (2001). Optimal Execution of Portfolio Transactions. *Journal of Risk*, 5-39.

Amihud, Y., & Goyenko, R. (2013, March). Mutual Fund's R2 as Predictor of Performance. *The Review of Financial Studies, 26*(3), 667-694.

Baquero, G., Horst, J. t., & Verbeek, M. (2005). Survival, Look-Ahead Bias, and Persistence in Hedge Fund Performance. *The Journal of Financial and Quantitative Analysis, 40*(3), 493-517.

Bellman, R. E. (1953). *An Introduction to the Theory of Dynamic Programming.* Santa Monica, CA: RAND Corporation.

Berk, J. B., & Green, R. C. (2004, December). Mutual Fund Flows and Performance in Rational Markets. *Journal of Political Economy, 112*(6), 1269-1295.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees.* Chapman and Hall/CRC.

Cao, C., Goldie, B. A., Liang, B., & Petrasek, L. (2016). What Is the Nature of Hedge Fund Manager Skills? Evidence from the Risk-Arbitrage Strategy. *Journal of Financial and Quantitative Analysis, 51*(3), 929-957.

Capponi, F., & Cont, R. (2019). Trade Duration, Volatility and Market Impact. *Available at SSRN: https://ssrn.com/abstract=3351736*.

Carhart, M. (1997). On Persistence in Mutual Fund Performance. *Journal of Finance, 52*(1), 57-82.

Chen, J., Hong, H., Huang, M., & Kubik, J. D. (2004). Does Fund Size Erode Mutual Fund Performance? The Role of Liquidity and Organization. *American Economic Review, 94*(5), 1276-1302.

Chevalier, J., & Ellison, G. (1997). Risk Taking by Mutual Funds as a Response to Incentives. *Journal of Political Economy, 105*(6), 1167-1200.

Cremers, K., & Petajisto, A. (2009). How Active Is Your Fund Manager? A New Measure That Predicts Performance. *Review of Financial Studies, 22*(9), 3329-3365.

Edwards, F. R., & Caglayan, M. O. (2001). Hedge Fund Performance and Manager Skill. *Journal of Futures Markets, 21*, 1003-1028.

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *The Annals of Statistics, 32*(2), 407-499.

Elton, E. J., Gruber, M. J., & Blake, C. R. (1996). Survivorship Bias and Mutual Fund Performance. *Review of Financial Studies, 9*(4), 1097-1120.

Elton, E. J., Gruber, M. J., Das, S., & Hlavka, M. (1993). Efficiency with Costly Information: A Reinterpretation of Evidence from Managed Portfolios. *The Review of Financial Studies, 6*(1), 1-22.

Evans, R. B. (2010). Mutual Fund Incubation. *Journal of Finance, 65*(4), 1581-1611.

Fama, E. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. *Journal of Finance, 25*(2), 383-417.

Fama, E. F., & French, K. R. (1992). The Cross-Section of Expected Stock Returns. *The Journal of Finance, 47*(2), 427-465.

Fama, E. F., & French, K. R. (1993). Common Risk Factors in the Returns on Stocks and Bonds. *Journal of Financial Economics, 33*(1), 3-56.

Fung, W., & Hsieh, D. A. (1997). Empirical characteristics of dynamic trading strategies: The case of hedge funds. *Review of Financial Studies, 10*, 275-302.

Fung, W., & Hsieh, D. A. (2000). Performance characteristics of hedge funds and CTA funds: Natural versus spurious biases. *Journal of Financial and Quantitative Analysis, 35*, 291–307.

Fung, W., & Hsieh, D. A. (2001). The Risk in Hedge Fund Strategies: Theory and Evidence from Trend Followers. *Review of Financial Studies, 14*, 313–41.

Fung, W., & Hsieh, D. A. (2002). Benchmarks of Hedge Fund Performance: Information Content and Measurement Biases. *Financial Analysts Journal, 58*, 22–34.

Fung, W., & Hsieh, D. A. (2004). Hedge fund benchmarks: A risk-based approach. *Financial Analysts Journal, 60*, 65–80.

Fung, W., & Hsieh, D. A. (2005). Extracting Portable Alphas from Equity Long/Short Hedge Funds. *Journal of Investment Management, 2*, 57-75.

Fung, W., Hsieh, D. A., Naik, N. Y., & Ramadorai, T. (2008). Hedge Funds: Performance, Risk, and Capital Formation. *Journal of Finance, 63*, 1777-1803.

Green, J., Hand, J. R., & Zhang, X. F. (2013). The Supraview of Return Predictive Signals. *Review of Accounting Studies, 18*(3), 692-730.

Grinblatt, M., & Titman, S. (1989). Mutual Fund Performance: An Analysis of Quarterly Portfolio Holdings. *The Journal of Business, 62*(3), 393-416.

Grinblatt, M., & Titman, S. (1992). The Persistence of Mutual Fund Performance. *The Journal of Finance, 47*(5), 1977-1984.

Gu, S., Kelly, B. T., & Dacheng, X. (2018). Empirical Asset Pricing Via Machine Learning. *NBER Working Paper No. w25398*.

Hendricks, D., & Wilcox, D. (2014). A Reinforcement Learning Extension to the Almgren-Chriss Framework for Optimal Trade Execution. *2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)* (pp. 457-464). London, UK: IEEE.

Hendricks, D., Patel, J., & Zeckhauser, R. (1993). Hot Hands in Mutual Funds: Short-Run Persistence of Relative Performance, 1974-1988. *Journal of Finance, 48*(1), 93-130.

Ibbotson, R. G., & Patel, A. K. (2002). Do Winners Repeat with Style? *Yale ICF Working Paper 00-70*.

Jagannathan, R., Malakhov, A., & Novikov, D. (2010). Do Hot Hands Exist among Hedge Fund Managers? An Empirical Evaluation. *Journal of Finance, 65*(1), 217-255.

Jensen, M. C. (1968). The Performance of Mutual Funds in the Period 1945-1964. *23*, 389–416.

Kacperczyk, M., Sialm, C., & Zheng, L. (2005). On the Industry Concentration of Actively Managed Equity Mutual Funds. *Journal of Finance, 60*(4), 1983-2011.

Kacperczyk, M., Sialm, C., & Zheng, L. (2008). Unobserved Actions of Mutual Funds. *Review of Financial Studies, 21*(6), 2379-2416.

Koh, F., Koh, W. T., & Teo, M. (2003). Asian hedge funds: Return persistence, style, and fund characteristics. *Working Paper*.

Liang, B. (1999). On the Performance of Hedge Funds. *Financial Analysts Journal, 55*, 72-85.

Lintner, J. (1965). The Valuation of Risk Assets and the Selection of Risky Investments in Stock Portfolios and Capital Budgets. *Review of Economics and Statistics, 47*(1), 13–37.

Lunde, A., & Timmermann, A. (2004). Duration Dependence in Stock Prices: An Analysis of Bull and Bear Markets. *Journal of Business & Economic Statistics, 22*(3), 253-273.

Mamaysky, H., Spiegel, M., & Zhang, H. (2007). Improved Forecasting of Mutual Fund Alphas and Betas. *Review of Finance, 11*(3), 359-400.

Nevmyvaka, Y., Feng, Y., & Kearns, M. (2006). Reinforcement learning for optimized trade execution. *Proceedings of the 23rd International Conference on Machine Learning* (pp. 673-680). New York, NY, United States: Association for Computing Machinery.

Pástor, Ľ., Stambaugh, R. F., & Taylor, L. A. (2015). Scale and skill in active management. *Journal of Financial Economics, 116*(1), 23-45.

Ritter, G. (2017). Machine Learning for Trading. *Risk*, 84-89.

Ritter, G., & Tran, M. (2018). Reinforcement Learning with Continuous States. *unpublished*.

Sharpe, W. F. (1966). Mutual Fund Performance. *The Journal of Business, 39*(1), 119-138.

Sirri, E. R., & Tufano, P. (1998). Costly Search and Mutual Fund Flows. *The Journal of Finance, 53*, 1589-1622.

Sun, Z., Wang, A. W., & Zheng, L. (2018). Only Winners in Tough Times Repeat: Hedge Fund Performance Persistence over Different Market Conditions. *Journal of Financial and Quantitative Analysis, 53*(5), 2199–2225.

Sun, Z., Zheng, L., & Wang, A. (2012). The Road Less Traveled: Strategy Distinctiveness and Hedge Fund Performance. *The Review of Financial Studies, 25*(1), 96–143.

Tibshirani, R. (1996). Regression Shrinkage and Selection via the LASSO. *Journal of the Royal Statistical Society, 58*(1), 267–88.

Tikhonov, A. N. (1963). Solution of Incorrectly Formulated Problems and the Regularization Method. *Soviet Mathematics, 4*, 1035-1038.

Welch, I., & Goyal, A. (2008). A Comprehensive Look at The Empirical Performance of Equity Premium Prediction. *Review of Financial Studies, 21*(4), 1455-1508.

Wermers, R. (2000). Mutual Fund Performance: An Empirical Decomposition into Stock-Picking Talent, Style, Transactions Costs, and Expenses. *The Journal of Finance, 55*(4), 1655-1695.

Zheng, L. (1999). Is Money Smart? A Study of Mutual Fund Investors' Fund Selection Ability. *Journal of Finance, 54*(3), 901-933.