

Action-based Feature Representation for Reverse Engineering Trading Strategies

A Thesis

Presented to
the faculty of the School of Engineering and Applied Science
University of Virginia

in partial fulfillment
of the requirements for the degree

Master of Science

by

Roy Hayes

May

2013

APPROVAL SHEET

The thesis
is submitted in partial fulfillment of the requirements
for the degree of
Master of Science


AUTHOR

The thesis has been read and approved by the examining committee:

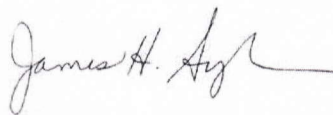
Peter Beling

Advisor

William Scherer

Stefano Grazioli

Accepted for the School of Engineering and Applied Science:



Dean, School of Engineering and Applied Science

May
2013

© Copyright April 2013

Roy Hayes Public

All rights reserved

Abstract

Stock trades now take place at unprecedented speeds. Aided by trading algorithms, market professionals can now buy and sell securities within milliseconds. These algorithms are capable of absorbing and then reacting to vast quantities of information, such as price movement, literally in the time it takes to blink an eye. Given this high speed, most trading algorithms do not take outside information, and in particular market news, into account. That is, trading algorithms are designed only to absorb endogenous data and thus tend to be deterministic. Furthermore, when provided with a constant or predictable type of information, these algorithms will always perform the same action.

To develop a model for incorporating variant trading algorithms, this study proposed to recover a subset of automated trading strategies derived from trade-level data. To determine the feasibility of this approach, we examined trading strategies used by participants during the Fall 2009 University of Virginia McIntire School of Commerce Hedge Fund Tournament. We determined that by using a variation of recursive partitioning it was in fact possible to recover trading strategies employed during the course of the tournament. This conclusion suggests that further research is warranted and provides justification for expanding the study to include trading strategies derived from real data.

Acknowledgments

I owe a lot of gratitude to many people for their support of my research. First and foremost, I would like to express the deepest gratitude to my advisor, Peter Beling, not only for his academic guidance but also for his wise counsel in both personal and professional matters. Professor Beling has directed my research toward interesting problems and innovative solutions, has encouraged challenging discussions, and has given me enough freedom to feel empowered by the journey. In particular, I would like to thank him for his time and patience in helping me to revise my writing: he has always been available to discuss my work even on weekends. He has never said anything negative about my research and in fact has always encouraged me to explore new ideas; he has provided the supportive atmosphere that has allowed my work to remain fresh. I also thank Professors William Scherer and Stefano Grazioli for serving on my thesis committee and for providing their valuable suggestions and insights. Last but certainly not least I would like to thank my friends and family for their continuing support.

Contents

1	Introduction	1
2	Trading and Return Replication: Static Method	5
3	Trading and Return Replication: Dynamics Methods	14
4	Inverse Reinforcement Learning	18
5	Apprenticeship Learning	26
5.1	Determining Which Security to Trade	27
5.1.1	Inverse Reinforcement Learning - Issues	28
5.1.2	The Need for Action Features	29
5.1.3	Supervised Learning Techniques - Recursive Partitioning . . .	31
5.2	Determining End of Day Delta Exposure	33
6	Empirical Tests and Results	36
6.1	McIntire Hedge Tournament	36
6.2	Results Determining Which Stock or Option Will Be Traded	39
6.3	Results Determining Delta Exposure	43
7	Conclusions	49

<i>Contents</i>	5
-----------------	---

8 Appendix	52
-------------------	-----------

8.1 State Space and Action Space Features	52
8.1.1 State Space Features - Security Selection Prediction	52
8.1.2 Action Space Features - Security Selection Prediction	53
8.1.3 State Space Features - Deleta Exposure Prediction	53
8.2 Delta Exposure Comparison	55
8.2.1 AAPL	55
8.2.2 AIG	55
8.2.3 C	56
8.2.4 DELL	56
8.2.5 DIS	57
8.2.6 GE	57
8.2.7 GOLD	58
8.2.8 GOOG	58
8.2.9 KO	59
8.2.10 MSFT	59
8.2.11 RIG	60
8.2.12 UBS	60

List of Figures

6.1	Delta Position Comparison	39
6.2	Security Selection Strategy	43
6.3	Training Set Delta Exposure Comparisons	44
6.4	GE's Hedging Strategy	45
6.5	DELL'S Full Tournament Delta Exposure Comparison	46
6.6	Tracking Error Comparison	47
6.7	Cumulative Tracking Error Difference Between Observed and Recovered Algorithms	48

List of Tables

4.1	Yang's IRL Actions	23
6.1	Action Identification Accuracy	41

Chapter 1

Introduction

Financial markets are complex environments comprised of many interrelated factors, including, perhaps most importantly, many different traders. Although these traders may share the goal of increasing capital, they often have—at least in the short term—partially independent objectives. Furthermore, even when traders seek to achieve the same objectives, they often accomplish them in drastically different ways. Nonetheless, these traders are interacting with one another, since the timing, volume, and price of each individual trade ultimately affect the broader market environment. These interactions of independent traders create the emergent properties known as stylized facts [1]. There are three such emergent properties: (1) no auto correlation of price returns, (2) auto correlation of absolute price change, and (3) aggregate normality of price returns over long time frames [2, 3, 4].

The need is for a viable, comprehensive model that will take into account the interactions of traders by reproducing these stylized facts. Such a model will help academic theorists, industry professionals, and regulators better understand financial markets. A comprehensive model will allow for identification of systemic risk, while also providing a test bed for regulatory reforms [2, 5]. There have been, in fact, many

different models that have reproduced these stylized facts; however, these models have been confined to academic applications because industry and government agencies have not found them reliable. This resistance has largely been due to the fact that available models have not sought to explain individual behaviors, such as the strategies employed by technical traders [2, 6, 7, 8, 3]. Instead, finance models have focused on mimicking actions by fitting a probability distribution. Trading strategies are not captured through this method—which means, for example, that regulators cannot determine how a new regulation will affect an individual trader.

There are several reasons that financial models have been designed to follow probability distributions. First, in the past academics did not have access to traders identities; all orders and trades were anonymous. This policy was in place as a means of safeguarding privacy; it also prevented other traders (as well as academic theorists) from reverse engineering any individual strategies. Moreover, up until recently orders have traditionally been placed by people. People are not perfectly rational [9, 10, 11]. Therefore, it is hard to recover an individual traders strategy with any degree of accuracy, because any such strategy will necessarily reflect human idiosyncrasy and, in some instances, error. Additionally, there is no guarantee that a person will use the same strategy in all market conditions [12, 13, 14].

This thesis examines the question of whether it is possible to determine a decision space relating to an individual trading strategy, such that there is a one-to-one relationship between a state and an action. Two key new factors have made determining a decision space plausible. First, the Commodity and Futures Trading Commission has changed its regulations so that identifying information regarding orders and trades is now accessible to academic researchers. Additionally, algorithmic traders now account

for 70% of the trades in equity markets and 35% in commodity markets. Algorithmic traders are computers that trade autonomously [15, 16]. For the purposes of this study, it is assumed that autonomous algorithms follow deterministic decision rules. In other words, the algorithm will perform the same action given that the algorithm finds itself in the same state. These two new factors increase the possibility of recovering strategies used by a majority of market participants. To date, however, very little research has been done in this area: this study aims to address that need by exploring methods for reverse engineering trading strategies.

As a case study, this thesis will examine algorithmic trading data from the McIntire School of Business Hedge Fund Tournament. The paper will provide a proof of concept by determining the decision space of the winning team and reproducing it in an out-of-sample test. The McIntire Hedge Fund Tournament is designed to teach students about electronic trading platforms and portfolio management. Each team is given a basket of stocks and options, which must be held for the full duration of the tournament. Teams are also provided cash, which they use to hedge their positions. The team that most closely follows a 1% annualized return over the entire tournament wins.

The McIntire School trading tournament is an ideal test case because all algorithms entered are deterministic in nature. Additionally, teams are limited to trading a relatively low number of securities, the objectives of the teams are clear, and all trades are captured with team identifiers. The challenge, however, is that teams may use a variety of approaches to reach their objectives, complicating the task of recovering individual team strategies. This thesis, then, is a feasibility study on whether it is possible to identify a single trading strategy in an environment where there are

a plethora of ways to reach the same objective.

The following two sections will provide a literature review of static and dynamic trading and return replication techniques, respectively. The paper will then review the machine-learning technique known as inverse reinforcement learning (IRL). IRL was the first technique used by this researcher. However, limitations in IRL led the researcher to use Recursive Partitioning. The methodology of recursive partitioning is examined in Section 5. An empirical test and results section follows. In the results section, both the testing procedure and results are presented. A more detailed explanation of the hedge tournament can also be found in this section. The paper then concludes with final thoughts.

Chapter 2

Trading and Return Replication: Static Method

Hedge fund cloning is a research topic that focuses on mimicking returns seen by hedge funds. The seminal paper on this subject was written by William Sharpe in 1992. Sharpe examined open-end mutual funds offered by Vanguard between 1985 and 1989. An open-end mutual fund is comprised of securities purchased with money contributed by many investors. Mutual funds have a manager or managers who buy and sell securities for the entire fund. These managers are mandated to meet or exceed returns of asset classes [17]. An asset class is a grouping of the same type of securities, such as corporate bonds. Below are the 12 asset classes Sharpe used in his study:

1. Treasury Bills – Salomon Brothers’ 90-day Treasury bill index;
2. Intermediate-term Government Bonds - Lehman Brothers Intermediate-term Government Bond Index
3. Long-term Government Bonds - Lehman Brothers Long-term Government Bond

Index

4. Corporate Bonds - Lehman Brothers Corporate Bond Index
5. Mortgage-Related Securities - Lehman Brothers Mortgage-Backed Securities Index
6. Large-Capitalization Value Stocks - Sharpe/BARRA Value Stock Index
7. Large-Capitalization Growth Stocks - Sharpe/BARRA Growth Stock Index
8. Medium-Capitalization Stocks - Sharpe/BARRA Medium Capitalization Stock Index
9. Small-Capitalization Stock - Sharpe/BARRA Small Capitalization Stock Index
10. Non-U.S. Bonds - Salomon Brothers Non-U.S. Government Bond Index
11. European Stocks - FTA Euro-Pacific Ex Japan Index
12. Japanese Stocks - DTA Japan Index

Sharpe's hypothesis was that a mutual fund's return can largely be explained by the average return of the asset classes to which the fund is exposed, and that therefore the returns of mutual funds should depend more on their relative exposure to a particular asset class and less on the specific securities they hold. To test this hypothesis, Sharpe examined monthly returns of mutual funds and attempted to find linear combinations of asset classes that best explained these returns. Below is the linear regression equation Sharpe used:

$$R_i = [b_1 F_{i1} + b_2 F_{i2} + \cdots + b_n F_{in}] + \epsilon_i,$$

where R_i represents the return estimated by the model for a specified month, F represents the return provided by each asset class during specified month, and b represents the relative weight placed on each asset class. In other words, $F_{1,1}$ represents the return on investment seen by treasury bills in the first month of the data set, and b_1 is the relative exposure a particular mutual fund has to treasury bills. Using the aforementioned equation, we can generate an estimate of the mutual fund's return for the first month in the data set (R_1). Note that $\sum_{j=1}^n b_{ij} = 1$, for all i [17]. The term ϵ_i represents the amount of return that is unexplained by this model. To determine how effective the model is at explaining the behavior of a particular mutual fund over several months, a coefficient of determination is calculated as

$$R^2 = 1 - \frac{Var(\epsilon_i)}{Var(R_i)}.$$

The closer to 1 the R^2 is, the better the model represents the data seen. Sharpes article shows results ranging from 92%-97%. In other words, Sharpes model was able to explain the returns of individual mutual funds by determining the relative weight of each asset class within each fund—that is, how much capital the fund invested in each asset class. The mutual funds Sharpe examined implemented a buy-and-hold strategy and were limited to little or no leverage. Given these parameters, Sharpes study clearly illustrates that the differences in the returns of mutual funds can be explained by the different weights each fund places on individual asset classes. This is easily demonstrated in Sharpes regression analysis [17]. However, other investment vehicles such as hedge funds are not bound by these restrictions [18]: hedge funds are able to carry leverage and can use a more dynamic trading strategy. As a result, Sharpes regression analysis does not work for these institutions [17].

In 1997, William Fung and David Hsieh revised Sharpes regression technique so that it was better suited for more dynamic trading strategies [18]. Fung and Hsieh took into account three critical additional factors: asset class (F), trading strategy, and leverage (w). Asset class, as in Sharpes paper, was defined as a grouping of the same type of securities. Trading strategy factored in whether the manager was long or short on the asset. Leverage referred to the quantity of the asset held. Leverage can be thought of as the weight placed on an asset, as in Sharpes paper. The one major difference, however, is that in Fung and Hsiehs model leverage was not constrained to sum to one but could range from - infinity to + infinity. Fung and Hsiehs model is as follows [18]:

$$R_t = \sum_{k=1}^N W_{kt} F_{kt} + \epsilon_t.$$

The advantage of this model is that it is more sensitive to trading techniques. It allows for $w = -1$, in the event of a hedge fund is shorting one contract, or $w = 2$, in the event that the hedge fund is long on two contracts. As in Sharpes study, this model was run over several months and a regression analysis was applied, resulting in a median coefficient of a determinant of 25%. The results are significantly lower than in Sharpes mutual fund analysis. Fung and Hsieh concluded that the dynamic strategies employed by hedge funds lower the ability of linear models to explain monthly returns [18]. Hedge funds change their portfolio allocation several times in a month, which is obscured in monthly returns.

Through the use of principal components, Fung and Hsieh discovered five general classes of hedge fund trading styles:

1. Systems/Opportunistic - technical driven traders, who also make bets on market events

2. Global/Macro - traders who only participate in the most liquid markets in the world (i.e., currencies and government bonds)
3. Value - traders who buy securities that are perceived to undervalued
4. Systems/Trend Following - technical driven traders, who tend to follow overall trend of the market
5. Distressed - traders who invest in companies near or recently emerging from bankruptcy

Fung and Hsieh attributed only 43% of the monthly return variances to these five trading styles [18]. However, even though these trading styles explain only a segment of the monthly return variances, they do suggest several key conclusions. For example, hedge funds that specialize in buying securities of companies that are undervalued are very sensitive to changes in the U.S. equity market.

In 1997, Brown and Goetzmann [19] classified Morningstar mutual funds into trading styles based on monthly returns. Unlike Fung and Hsiehs classifications, however, Brown and Goetzmans study did not assume a linear model. As Fung and Hsieh illustrated, a linear models predictive power degrades if the mutual fund has a dynamic strategy. To better model dynamic strategies, Brown and Goetzmann allowed the weights on asset classes to vary through time. Their model is illustrated below:

$$R_i = [b_{t1}F_{t1} + b_{t2}F_{i2} + \cdots + b_{Tn}F_{in}] + b_0 + \epsilon_i.$$

Brown and Goetzmann used a regression model identical to Sharpes model except b_{i1} is allowed to change through time. If we assume the asset classes are the same as the Sharpe model, b_{i1} represents the fund's exposure to treasury bills during the first

month in the data set. Additionally, b_0 is a constant that is specific to each strategy. To estimate b_{in} and to classify the mutual funds into groups, the following procedure was run:

1. Specify the number of trading style classes (K)
2. Specify the time period of classification (T)
3. Specify the number of trading funds (N)
4. for all combinations N funds divided into K classes complete the following calculation:

$$\bar{R}'_t = \sum_{i=1}^N \frac{R_{it}}{\text{var}(R_{it} - \bar{R}_t)} \text{sum}_{i=1}^N \text{var}(R_{it} - \bar{R}_t)$$

5. Select combination that provides:

$$\min \sum_{j=1}^K \sum_{t=1}^T \sum_{i=1}^N (R_{ti} - \bar{R}'_{tj})^2$$

for all funds (i) in class (j)

6. Run regression one each class treating individual months as independent data sets, estimate b_{Tn} for each.

Brown and Goetzmann were able to derive trading styles and state meaningful interpretations of the results. For example, a trading classed named “Global Timing” dynamically increases and decreases its exposure to the U.S. market [19]. However, Brown and Goetzmans method did not improve upon Fung and Hsiehs principal component method, explaining only 37% of the monthly return variation.

Using different hedge fund data, Brian Liang performed the same regression model that Fung and Hsieh developed. Langs findings were consistent with Fung and Hsiehs

results. However, he interpreted B_0 , also known as the intercept term, as the managers contribution to the returns [20]. Liang found that the intercept term contributes between -5% and 1% for each asset group. In other words, hedge fund managers on average provide only a marginal improvement, if any, over asset class returns.

In 2007, Hasanhodzic and Lo developed a regression model similar to Fung and Hsiehs model. The only significant difference is that Hasanhodzic and Lo limited their regression model to five asset classes, each of which had exchange traded fund (ETF) counterparts. The goal of this analysis was to provide linear combinations of ETFs that the average investor could use to gain returns similar to those of hedge funds. Additionally, Hasanhodzic and Lo claimed this approach would prevent the overfitting seen in Fung and Hsiehs principal component/regression approach. Overfitting occurs when a model excels in a training phase but performs poorly in out of sample test. The five ETF classes used in this regression model are as follows [21]:

1. US Dollar Index
2. BOND: Lehman Corporate AA Intermediate Bond Index
3. CREDIT: Lehman BAA Corporate Bond Index
4. S&P 500 index
5. Goldman Sachs Commodity Index

Using a 24-month rolling window to avoid a look-ahead bias, Hasanhodzic and Lo illustrated that they were able to develop a linear combination ETF strategy that had higher returns than the market. However, the linear clones did not perform as well as the hedge fund counterparts [21]. This conclusion again reinforces Fung and Hsiehs

finding that the dynamics of hedge fund strategies are obscured by the aggregation of data into monthly returns.

Merrill Lynch (Merrill Lynch Factor Index) and Goldman Sachs (Absolute Return Tracker Index) have both created indices that are supposed to mimic hedge fund returns. In 2010, Meyfredi et al., spurred on by these hedge fund replication indices, applied Kalman filtering to their own hedge fund replication. Kalman filtering uses a regression equation similar to Brown and Goetzmanns. The significant difference is that Kalman filtering requires at each time step, B_t (weight placed on the asset class) is estimate from B_{t-1} using a transition matrix. Therefore, not only does B have to be estimated the transition matrix A also has to be estimated. In the equation developed by Meyfredi et al., n and ϵ are normally distributed error terms with mean equal to 0 [22]:

$$B_t = AB_{t-1} + n_t$$

$$R_t = B_t'F + \epsilon_t.$$

Kalman Filter model gives the best R^2 value ranging from [62%, 80%] when the nonlinear models are applied to the same 24-month rolling window employed by Hasanhodzic and Lo. This indicates that the Kalman filter is capable of recovering a larger amount of the dynamic hedge fund trading strategy. However, when applied to out-of-sample data, the Kalman filter provided an average annualized excess return of 0.04, which is significantly lower than the returns of the original hedge funds [22]. Excess returns are returns that exceed a market benchmark, such as the S&P 500. This indicates either that the Kalman filter overfit the data or that hedge funds do not carry out the same strategy from month to month.

As the preceding review suggests, for the past two decades, academics, individuals, and institutions have tried to reverse engineer successful strategies. By mimicking these strategies, investors can achieve similar returns without incurring the upfront cost of research. Recently, subscription services have begun to offer investors the opportunity to pay for the privilege of automatically having their accounts mimic successful traders. This has allowed subscribers to see all of the trades associated with a profitable trading strategy. The proposed methodology in this thesis can be applied to the subscription service data with the hopes of better understanding real-world trading strategies. The next section will discuss some of these techniques in greater detail.

Chapter 3

Trading and Retrun Replication: Dynamics Methods

Long-term securities, such as stocks, can be used to replicate complex instruments. The original example is when the purchase of a stock through riskless borrowing is used to replicate the payoff curve of an option on said stock. If volatility is known, the Black-Scholes formula can be used to exactly replicate an option.

An option gives the holder the right but not the obligation to buy (call option) or sell (put option) a stock at a specified price (K). The duplication of an option price is a simple form of trading strategy replication. To replicate the strategy, the Black-Scholes formula is used to calculate Delta (Δ). Delta measures the option price change when the underlying stock increases by one dollar. The formula for delta is given below:

$$d_1 = \frac{\ln(\frac{S}{K}) + (r - \delta + .5\sigma^2)t}{\sigma\sqrt{t}},$$
$$\Delta_{Call} = e^{-\delta T} N(d_1),$$

$$\Delta_{Put} = e^{-\delta T} N(d_1) - 1.$$

Using the current price of the stock (S), volatility of the stock (σ), the dividends the stock pays (δ), the strike price of the option (K), and the time till the option expires (t) a relationship between the option price movement and the stock price movement can be established. The function $N(x)$ is the cumulative standard normal distribution function; it gives the probability that a number randomly drawn from a standard normal distribution falls below x [23]. If Δ and σ are assumed to be constant, it becomes trivial to calculate the payoff curve of an option:

Buy Δ amount of stocks,

Borrow $Ke^{-rt}N(d_1 - \sigma\sqrt{t})$ amount of cash.

The problem with the Black-Scholes approach is that Δ changes as the stock price changes. Therefore, delta must be recalculated at every time step, and a dynamic strategy of buying and selling the stock must be used to replicate the option. Dynamic replication of options is a branch of research that seeks to determine the optimal way of replicating an option payoff curve. When simplifying assumptions—such as σ is constant and the market is always open—are removed, the problem becomes nontrivial. In fact, perfect replications are impossible in real-world markets [24].

Assume you are given an option with a payoff curve of $F(P_T, Z_T)$, where P_T is the price of the underlying asset at time T and Z_T is a vector of state variable which dictate the value off the option at time T . The value of the option does not correlate perfectly to P_T , thus making the stated dynamic strategy no longer optimal. For specific examples where the aforementioned correlation is not exact, see Bertismas et al. The objective of dynamic replication is to find a combination of stock shares and borrowed cash to $\min E[V_T - F(P_T, Z_T)]^2$. In other words, the objective is to

minimize the square difference between the portfolio value (V_T) and the option value at time T . Therefore, a measure of how close the options payoff curve is replicated is the squared error, as shown in the following equation:

$$\epsilon = \sqrt{\text{Min}_{\theta_T} E[(V_T - F(P_T, Z_T))^2]}.$$

Bertismas et al. impose the constraint that the portfolio be self-financed. Aside from the initial capital necessary to establish the first position, all changes in cash (B) being borrowed and stocks (θ) being held must be financed by the buying and selling of stocks. The equation below illustrates the constraint:

$$P_{t_{i+1}}(\theta_{t_{i+1}} - \theta_{t_i}) + B_{t_{i+1}} - B_{t_i} = 0,$$

Which implies:

$$V_{t_{i+1}} - V_{t_i} = \theta_{t_{i+1}}(P_{t_{i+1}} - P_{t_i}).$$

Assuming that the expected return divided by variance for any period is bounded, there exists an optimal replication strategy that gives the lowest possible ϵ [24]. Dynamic programming is used to work backward from the terminal payoff states, thereby identifying the optimal strategy. Kat and Palaro extend this process to replicating a fund of funds [25]. A fund of funds is a hedge fund that is made up of smaller hedge funds. Kat and Palaro attempt to recover the risk profile of observed hedge funds (i.e., the Sharpe ratio profile). Kat and Palaro use a single asset that can be traded to replicate hedge fund returns. Using their replication strategy, Kat and Palaro were able to get within 10% of the observed hedge fund returns [25].

The problem with Kat and Palaros methodology is that it assumes that a single

asset can be used to replicate a hedge funds risk profile. This single asset, however, may not be identifiable; even if it is, it may not be adequate to mimic the hedge fund as a whole. In many cases, individuals and organizations employing Kat and Palaros methodology end up basing their results on a series of actions, such as multiple trades, that taken together represent a strategy—rather than on the movement of a single asset.

Expanding on dynamic programming a company named Adaptrade Software claims to be able to replicate strategies by using past trades. This software examines past trades and their outcomes, such as profits/losses occurring from a trade. Using a genetic algorithm, the software combines technical trading rules (i.e., moving average, relative strength index, etc.) to generate a strategy that mimics the observed trades and outcomes. In its brochure the company claims that its clone strategy generated profits that were 3% lower than the observed strategy. It is important to note that Adaptrade does not claim the observed strategy follows the same rules as the clone strategy. In other words, Adaptrade has not claimed to have found a trading strategy decision space. Additionally, it has not published any out-of-sample testing. However, the fact that a commercial company has created reverse-engineered trading software suggests that there is a desire for a methodology to reverse engineer trading decisions. The following section will examine a machine-learning technique called inverse reinforcement learning, which has been applied to trading strategy classification.

Chapter 4

Inverse Reinforcement Learning

The following several paragraphs are used with permission, given by Peter Beling [26]. Imitation learning is a subfield of machine learning in which the objective is to learn to mimic an agent's behavior solely through observation of the agent's actions. Technical approaches to imitation learning generally fall into two broad categories. *Behavioral cloning* approaches attempt to predict actions directly from an observed feature vector that describes the environment. In *inverse reinforcement learning (IRL)*, by contrast, training examples take the form of trajectories through the feature space defined in terms of an underlying model of decision task and environment as a control or sequential optimization problem. IRL algorithms, which are set in the context of a sequential optimization framework known as a Markov decision process (MDP), attempt to discover the objective or reward function for the underlying sequential decision problem solely on the basis of observing the decision makers' solution to that problem. This approach is appealing in imitation learning because knowledge of the reward function offer the promise that behavior can be predicted in domains unseen during training.

A finite-state, infinite horizon *Markov decision process (MDP)* is defined as a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, r)$, where $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ is a set of n states; $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$ is a set of m actions; $\mathcal{P} = \{P_a\}_{a=1}^m$ is a set of state transition probabilities; γ is a discount factor; and r is a n -dimensional vector such that r_s is the reward experienced on reaching state s . For any $a \in \mathcal{A}$ and P_a is a $n \times n$ matrix, each row of which, denoted as P_{as} , is the transition probabilities upon taking action a in state s .

Consider a decision maker that chooses actions according to a *policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maps states to actions. Define the *value function* at state s with respect to policy π to be $V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_{s^t} | \pi]$, where the expectation is over the distribution of the state sequence $\{s^0, s^1, \dots\}$ given policy π , where superscripts index time. A decision maker who aims to maximize expected reward will, at every state s , choose the action that maximizes $V^\pi(s)$. Similarly, define the *Q-factor* for state s and action a under policy π , $Q^\pi(s, a)$, to be the expected return from state s , taking action a and thereafter following policy π . Given a policy π , $\forall s \in \mathcal{S}, a \in \mathcal{A}$, $V^\pi(s)$ and $Q^\pi(s, a)$ satisfy $V^\pi(s) = r_s + \gamma \sum_{s'} P_{\pi(s)s}(s') V^\pi(s')$ and $Q^\pi(s, a) = r_s + \gamma \sum_{s'} P_{as}(s') V^\pi(s')$. The well-known Bellman optimality conditions state that π is optimal if and only if, $\forall s \in \mathcal{S}$, we have $\pi(s) \in \arg \max_{a \in \mathcal{A}} Q^\pi(s, a)$ [27].

Given an MDP $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, r)$, let us define the *inverse Markov decision process (IMDP)* $M_I = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, \mathcal{O})$. The process M_I includes the states, actions, and dynamics of M , but lacks a specification of the reward vector, r . By way of compensation, M_I includes a set of observations \mathcal{O} that consists of state-action pairs generated through the observation of a decision maker. We can define the *inverse reinforcement learning (IRL)* problem associated with $M_I = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, \mathcal{O})$ to be that

of finding a reward vector r such that the observations \mathcal{O} could have come from an optimal policy for $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, r)$. The IRL problem is, in general, highly underspecified, which has led researchers to consider various models for restricting the set of reward vectors under consideration. Ng and Russel [13], in a seminal consideration of IMDPs and associated IRL problems, observed that, by the optimality equations, the only reward vectors consistent with an optimal policy π are those that satisfy the set of inequalities $(P_\pi - P_a)(I_n - \gamma P_\pi)^{-1}r \geq \mathbf{0}, \forall a \in \mathcal{A}$, where P_π is the transition probability matrix relating to observed policy π and P_a denotes the transition probability matrix for other actions. Note that the trivial solution $r = 0$ satisfies these constraints, which highlights the underspecified nature of the problem and the need for reward selection mechanisms. Ng and Russel [13] advance the idea choosing the reward function to maximize the difference between the optimal and suboptimal policies, which can be done using a linear programming formulation. In the sections that follow, we propose the idea of selecting reward on the basis of MAP estimation in a Bayesian framework.

A principal motivation for considering IRL problems is the idea of apprenticeship learning, in which observations of state-action pairs are used to learn the policies followed by experts for the purpose of mimicking or cloning behavior. By its nature, apprenticeship learning can lead to problems in situations where it is not possible or desirable to observe all state-action pairs—that is, in situations where some state-action pairs are not applicable to the decision makers policy. In recent approaches to apprenticeship learning, partial policy observation is dealt with by searching mixed solutions in a space of learned policies with the goal that the accumulative feature expectation is near that of the expert [12, 28]. In such approaches, the reward func-

tion is approximated by a linear combination of features, which in turn allows for linear approximation of value functions with consequent simplification of the learning problem. In such methods, algorithm performance is strongly influenced by the modelers choice of features. Another algorithm for IRL is policy matching in which the loss function penalizing deviations from an experts policy is minimized by tuning the parameters of reward functions [29]. Other approaches to IRL include game-theoretic methods [28] and algorithms derived from linearly solvable stochastic optimal control [30].

IRL has been applied to a number of problems, most related to the problem of learning from demonstrations. Apprenticeship-learning algorithms based on IRL—which leverage expert demonstrations to efficiently learn controls for tasks being demonstrated by an expert—have been applied to automatic control of helicopter flight [31] and modeling of driver route preferences [32].

Steve Yang et al. applied IRL to the problem of classifying simulated trading strategies. They addressed the following question: given N number of trading strategies, is it possible to cluster these strategies in a single reward space? The trading strategies they examined placed orders into an electronic limit order book [14]. An order specifies three things:

1. Side - whether it is a buy or sell order
2. Quantity - how many contracts or shares are desired
3. Price - the price the order is willing to trade at. This can be an actual price or the designation that the order is willing to transact at any price

Orders that do not have a corresponding order to transact with at the specified

price are called limit orders. Limit orders are placed into a queue until a corresponding order is found. In Yang et al. traders can perform the following actions:

Action Description
PBH - place buy order higher than the 3rd best bid price
PBL - place buy order lower than the 3rd best bid price
PSH - place sell order higher than the 3rd best ask price
PSL - place sell order lower than the 3rd best ask price
CBH - cancel buy order higher than the 3rd best bid price
CBL - cancel buy order lower than the 3rd best bid price
CSH - cancel buy order higher than the 3rd best ask price
CSL - cancel sell order lower than the 3rd best ask price
TBH - trade buy order higher than the 3rd best bid price
TBL - trade buy order lower than the 3rd best bid price
TSH - trade sell order higher than the 3rd best ask price
TSL - trade sell order lower than the 3rd best ask price

Table 4.1: Yang's IRL Actions

Each trader may perform different actions depending on what state he or she is in. However, the underlying assumption of IRL is that traders will most likely choose the action in each state that leads to the highest possible reward. Therefore, by defining a function that describes the reward for each state it becomes possible to identify the optimal solution. Yang et al. defined a state space based on order imbalance and on the inventory position of the trader. The state space features are as follows:

TIM - volume imbalance at the best bid/ask $[-1,0,1]$.

NIM - volume imbalance at the 3rd best bids/ask $[-1,0,1]$.

POS - inventory of the trader $[-1,0,1]$.

Each state has only three possible values, giving 27 potential states [14]. A -1

means the sell side has more limit orders than the buy side; in the case of inventory, it means that the trader is short a large amount of shares. A 0 means there is very little imbalance and the trader owns very little shares. Positive 1 is the reverse of the -1 state. Given a set of actions by a trader, a reward function can be generated that maximizes the difference in reward between the observed policy and any other potential policy.

Yang et al. generated these reward functions for simulated trading strategies running in an artificial market. The reward function characterized how much an expert valued entering a specific state. The goal was to determine if high-frequency traders could be distinguished from market makers and opportunistic traders. Using a linear discriminant analysis, Yang et al. demonstrated that it is possible to obtain a 95% classification accuracy between high frequency traders and the market maker and opportunistic traders [14]. However, when market makers were compared to opportunistic traders, the accuracy fell to 70%.

A followup study presented by Yang et al. in [33]. suggests that a Gaussian-based IRL provides better classification results than those modeled in their earlier study. When asked to identify trader A versus trader B, the linear IRL achieved 60%, where Gaussian received an accuracy of 96.5%. However, although Yang et al. illustrate that IRL is useful in classify trading strategies, they were unable to return policies that mimic actual trading strategies. IRL works under the assumption that the state space is already known. In the case of Yang et al., a state space is hypothesized and the observed actions are mapped to that state. However, this leads to the problem of different actions occurring in the same state (i.e., Yangs assumption of nondeterministic trading strategies). Algorithms follow a deterministic strategy (A_i

is always chosen in S_i); the only way this occurs is if the strategy is nonstationary (changes over time) or the state space does not represent the algorithms decision space. In other words, the decision space takes other variables into account which are not in the state space description. Given the short time window in which the present study is performed, we will hypothesize that the state space does not represent the decision space.

Chapter 5

Apprenticeship Learning

Recovering a trading strategy has many elements of a classical apprenticeship learning problem. A trading strategy is a set of decision rules that lead to the selection of a security, as well as, a decision to buy or sell the security and the amount of shares that will be traded. The apprentice is given a set of discrete time events to learn from. All security prices and market history are available to the observer. Additionally, the financial position of the trader is known (i.e., inventory, cash, profit, losses, etc.). The apprenticeship problem is to predict trader actions in future time periods.

To examine this problem, I used data from the McIntire Hedge Fund Tournament. The goal of the tournament is for participants to hedge a portfolio of illiquid assets by buying an offsetting combination of stocks and options. The winner of the tournament is the team that comes the closest to making a 1% annualized return with their combined illiquid and liquid portfolio. In essence, the objective of the tournament is for teams to limit the variance of their portfolios. I chose to recover the winning team's trading strategy because they employed an algorithm that did not make any erroneous trades, such as selling a stock when none existed in inventory.

The winning team used a delta hedging strategy. This trading strategy involved making two decisions when hedging the portfolio. The algorithm first needed to choose which security to trade, and then had to determine the desired end-of-day delta exposure. The desired delta exposure determines the direction (i.e., taking a long versus a short position) as well as the volume (i.e., how many shares) are traded. Each decision is treated as an independent problem. The algorithm decisions are treated as two independent problems because this approach lowers the complexity of recovering the decision space. Instead of simultaneously solving for two independent variables this methodology allows for solving for one variable at a time. The following section will explain the methods used to determine which security the algorithm would choose to trade.

5.1 Determining Which Security to Trade

The trading algorithm has a set of options that are available to trade. The options have different expiration prices and therefore different monetary valuations. For this section we make the simplifying assumption that at the end of day the portfolio has a delta exposure of 0. In other words, every trade that is taken leads to a delta exposure of 0. We hypothesize that the observed algorithm is attempting to complete this task. Based on this hypothesis, we assume that the recovered algorithm will accomplish the goal of maintaining a delta neutral portfolio. The first machine-learning technique that was applied to the data was IRL.

5.1.1 Inverse Reinforcement Learning - Issues

IRL was applied to the tournament data because of the success that was seen in Yang et. al [33]. However, the results in our application of this methodology were very poor. It is interesting to examine why these results failed, when Yang et. al. experienced such positive results. A natural reward function is the difference in delta exposure of the illiquid and liquid portfolios. However, the assumption of a 0 end-of-day delta exposure creates a unique formulation. Experts will take an action where their expected reward is 0. This is starkly different from the traditional IRL problem, where experts are assumed to maximize their expected rewards. Additionally, there are multiple actions (i.e., multiple options) that can be taken to lower overall delta exposure to 0. Thus, the naive state space IRL is not applicable to this problem.

IRL algorithms are designed to solve sequential decision problems, in essence trading off short-term rewards for long-term benefits. However, this assumes that a transition matrix is known to the apprentice; in other words, the trading algorithm would be able to predict future price movements. It is unlikely that the students in the tournament were able to develop an accurate forecasting model. Furthermore, the point of the contest was to hedge against unknown future states—that is, to design an algorithm that would maintain a predictable return regardless of the future state of the market. Therefore, assuming that each day teams were presented with a new problem, they needed to determine—for that day—the best way to limit their portfolio exposure. In other words, in the tournament, every day is treated as an independent problem, not a sequential decision problem. Therefore, IRL is not a viable tool for recovering trading strategies in the tournament. Instead, then, we determined that a supervised learning approach is a preferable approach for recovering the strategy the

winning team used to acquire a given option.

5.1.2 The Need for Action Features

The trading algorithm selects an option to trade, but all actions (trades), under the aforementioned assumptions, lead to a delta exposure of 0. It is our hypothesis that the trading algorithm has larger-level objectives that it is trying to meet, such as maximizing delta change per dollar. Therefore, to accurately reverse engineer trading strategies it becomes necessary to discover action features that facilitate these higher-level objectives. For example, option A may be worth \$1.00, while option B may be worth \$0.50. Although both options can be used to delta hedge a portfolio, an algorithm that attempts to maximize delta change per dollar may select option B over option A because option B has a higher delta-to-dollar ratio. This difference could lead the algorithm to use option B when hedging the portfolio.

It is important to mention that this is starkly different from traditional MDPs, upon which IRL and other supervised learning techniques are based. In an MDP model, decisions are made based on the likelihood of arriving at future states. It is conceivable that the information provided in action features could be represented in a large state space containing all possible option prices and cash levels. Additionally, given a large enough number of observations it is possible that patterns could be discerned from the trajectory through the previously mentioned state space. However, the large state space and relatively small number of observations in the training set (approximately 45 trading days) led us to use a state space representation employing action features. Supervised learning traditionally assumes that actions are chosen based on their probability of transitioning to a desired state. However, due to the

aforementioned reasons, we assumed that the actions were chosen based on a comparison to other actions. Features are given to each potential trade. Action features are indicator variables $(1, 0)$ that describe potential actions. If a subset of indicator variables are found to be always on or off in actions that are chosen, and the reverse is the case for actions that are not chosen, then we conclude that these features are used to select which security to trade. The following features were applied to the tournament data:

1. Security is an option
2. Security expires in May
3. Security is already own
4. Lowest price option
5. Lowest absolute value trade
6. Largest delta/price ratio
7. Largest short
8. Sign switch

For a detailed explanation of the features for both the action space and the state space, see the appendix. The results of this method will be presented and explained in the empirical results section. However, I will note that certain features, such as (1) “Security is an option” and (2) “Security expires in May,” were found in all selected actions. In other words, the team’s strategy was to hedge using only options that expired in May.

5.1.3 Supervised Learning Techniques - Recursive Partitioning

An open source version of classification and regression trees, was used to determine the trading algorithm. A superset of all possible actions (trades) \mathcal{A} was made for each trading day. A trade is marked positive if the algorithm implemented the trade on that day; otherwise it was marked as negative. Additionally, each action had a set of features (F_{ai}) that described the action, and each trading day had a set of features (F_{sj}) that described the current market conditions. Our objective in employing recursive partitioning was to use the set of features to predict which trades would be marked as positive and which would be marked as negative.

Recursive partitioning results in a binary tree. A binary tree consists of one root (origin) node that splits the data into two groups based on a specified condition. The method for building binary trees begins with selecting a single feature that best splits the data into two groups. (In this context, the term *best* refers to the split that maximizes the separation between classes.) The subgroups are then split into smaller groups, and they in turn are again split into yet smaller groups, in a recursive process. This continues until each class is completely classified or until a minimum observation threshold is reached in each node [34]. Below is the equation used to determine which feature to use as the partition rule:

$$\max[P(L)P(C_1) + P(R)P(C_2)].$$

The proportion of observations that is classified to the left branch of the tree is represented by $P(L)$ and the proportion to the right is represented by $P(R)$. The proportion of observations on the left branch classified as category 1 is denoted as

$P(C_1)$, while the proportion of observations on the right branch classified as category 2 [34] is denoted as $P(C_2)$.

Although traditional recursive partition can provide meaningful insight into the trading strategies found in the tournament, it cannot correctly classify those strategies. This is because it presents all observations as one large set. However, this was not how the trading algorithm used by the team arrived at decisions. Each day the trading algorithm presented its current state and potential actions. The algorithm selected one action, thereby categorizing all other possible actions as negative. In theory, the recursive partitioning algorithm should follow the same pattern; in other words, it should determine a partition or a set of partitions that labels one action positive and the rest negative for each day. If the partitions are not generated in this manner, misclassification can occur; specifically, it can result in multiple actions being labeled positive for a specific day. To safeguard against such a misclassification, we extended recursive partitioning by grouping the observations by trading day. This was a significant and indeed an unorthodox step, because tree-style classifications do not typically include a temporal component.

The modified partitioning split we employed sought to maximize the number of days correctly classified. We assumed that a day was correctly classified when all actions in that day were assigned the correct label. As in traditional recursive partitioning, the first partition was generated by selecting a single feature that best split the data. We continued this process until the data was correctly classified or a minimum threshold was reached. The results section will illustrate that it is necessary for actions to contain features; it will also demonstrate that the temporal relationship of the data must be included as a factor when trying to recover a trading strategy.

5.2 Determining End of Day Delta Exposure

When recovering a trading strategy, there are a few basic functions that must be determined. First, it is essential to identify what will trigger the strategy—that is, what enables the strategy to become active; then, once the strategy is active, it is necessary to ascertain which security will be traded. (The preceding sections described techniques for accomplishing this task.)

The final part of the strategy involves determining whether to buy or sell shares and how many shares to trade. In this section, the previous assumption that all trades lead to a delta exposure of 0 is dropped. Instead, the machine-learning technique known as regression trees is used to predict the delta exposure at the end of the day. Using the Black-Scholes formula, the desired end-of-day delta exposure, and the asset being traded, it becomes trivial to determine not only the trade direction (i.e., buy or sell) but also the desired quantity to be traded. Therefore, in this section we will predict desired end-of-day delta exposure as a means of determining both the direction of a trade and the quantity being traded.

As with recursive partitioning, regression trees first try to classify states into classes. The apprenticeship learning algorithm is presented as a set of state features (F_{sj}). For the purposes of this study, we assume that the option being traded is known, having been arrived at using the methodology discussed in the previous section. Therefore, intrinsic action values such as price of a trade are treated as state features in the method we examine here. These features represent the current state at the beginning of the day. Additionally, the learning algorithm is given the dependent variable delta exposure (D_j) at the end of the day. Using the state features, the learning algorithm is trained to predict the end-of-day delta exposure.

The mean value of the dependent variables (delta exposure of trades) that are partitioned into the same class is used as the prediction for that class. The splitting criteria seek to maximize the between-groups sum of squares [34], as follows:

$$\max SST - (SSL + SSR)$$

Where

$$SST = \sum (y_i - \bar{y}).$$

In the above equation, SSL is the sum of squares for the left node and SSR is the sum of squares for the right node. The algorithm terminates if it cannot generate a minimum distance between the sum-of-square error of the children versus the parent node. Using the mean value of the dependent variable gives one possible prediction for a potentially large number of observations. Although this makes the regression trees tractable, it limits their predictive power. Therefore, in this thesis the method of regression trees was augmented to use stepwise linear regression instead of the mean value as the final node's prediction. Stepwise linear regression adds and removes variables in a standard linear regression model. The goal is to provide a linear model that has the highest adjusted R^2 value. For all potential variables a simple linear regression model is fitted. The t^* statistic is taken for each model to test whether or not the slope is 0. Below is the equation for the t^* statistic:

$$t^* = \frac{b_1}{MSE / \sum (X_i - \bar{X})^2}.$$

The X variable with the largest t^* value that also has a p-value less than 0.15 is added into the model [35]. Assuming that the "delta of the option" had the largest t^* value and had a p-value less than 0.15, then it would be added into the model. In

the next iteration, all models will contain "delta of the option," and all other possible variables will be added to the model one at a time. A t^* statistic is taken of each one, and the process repeats. If there are two or more variables in the model, the procedure changes: all variables in the model have their t^* statistic taken and p-values calculated. If a variable has a p-value greater than 0.15, then it is removed from the model, after which the aforementioned process of adding variables is applied again [35]. This process continues until no more variables can enter or leave the model. Stepwise regression allows for a closer prediction of the response variables. However, it also creates the possibility of overfitting to the data. Overfitting can lead to erratic behavior of the model during out-of-sample testing. In the next section, both versions of regression trees will be used and their performance compared to one another.

Chapter 6

Empirical Tests and Results

6.1 McIntire Hedge Tournament

The McIntire Hedge Fund Tournament is held semiannually at the University of Virginia. The goal of the hedge fund tournament is to hedge a portfolio of illiquid assets using stocks and options, in such a manner that the portfolio provides a 1% annualized return throughout the tournament. All trading must be done electronically, which allows for the capture of every trade in the tournament. It is important to note that teams are assumed to be too small to change the price of the asset they are trading.

The tournament is held over approximately three hours continuous time period. Each minute is equivalent to a simulated day, with a bid and ask price being presented for each asset. Teams possess a basket of stocks and options that they are not allowed to trade during the tournament. With \$16million in cash, teams must take offsetting positions to limit the volatility of their overall portfolio. Additionally, teams must try to consistently deliver a 1% annualized return. A measurement known as the tracking error (TE) is taken every Sunday. The team with the lowest cumulative

tracking error is crowned the winner. Below is the equation for the tracking error:

if Portfolio Value \geq Target:

$$TE = (PV - T)/2.$$

Else:

$$TE = (T - PV).$$

The target (T) in the equation above is the value of the initial portfolio plus a 1% annualized return to date. To minimize tracking errors, teams enter long or short positions in stocks or options to offset the illiquid position. Five rules limit the ability of teams to enter positions:

1. Margin account value must not exceed \$22 million dollars
2. Must have at least 30% of the margin account value in cash at all times
3. Can not trade a specific asset more than once a day
4. For a trade to be initiated the required amount of cash must be present in the portfolio
5. Can not trade any asset that has a value of zero dollars

Assets are broken up into families. Twelve asset families make up the tradeable world for the tournament. These asset families are derived from the following twelve stocks:

1. Apple
2. AIG

3. Citigroup
4. Dell
5. Walt Disney
6. General Electric
7. Gold
8. Google
9. The Coca-Cola
10. Microsoft
11. Transocean
12. UBS

A stocks value throughout the tournament is tuned to the historical volatility and trend of the stock from the previous six months. Through the use of Brownian motion, a price path is produced that is similar to historical data but that does not exactly replicate it. Using the Black-Scholes equation, the price for each option is calculated. There are two sets of options: the first expires in March, and the second expires in May. Additionally, there are five puts and calls for each security at each expiration month; the options have varying strike prices. This results in 252 securities that can be traded during the tournament. However, some securities are placed into the illiquid portfolio, lowering the number of tradeable securities to 221.

Twelve teams competed in the tournament. However, only three teams were eligible to have their strategies recovered. The majority of the teams made a large

number of errors when trading, resulting in large tracking errors, after which these teams stopped trading. As a result, there was not enough information to recover the trading strategies used by these teams. Additionally, some teams traded manually. Manual trading adds another layer of uncertainty. The assumption in recovering algorithmic trading strategies is that algorithms presented with the same state information will make the same decision every time. Since this study involved a proof of concept, one team in particular, the winning team, was used to illustrate the results.

6.2 Results Determining Which Stock or Option Will Be Traded

The majority of the teams used delta hedging to accomplish the goal of variance reduction. This can be seen by comparing the delta exposure of each team's strategy to that of a non-hedged portfolio. To determine which asset would be chosen to trade, we assumed, for simplicity, that all trades resulted in a delta exposure of 0.

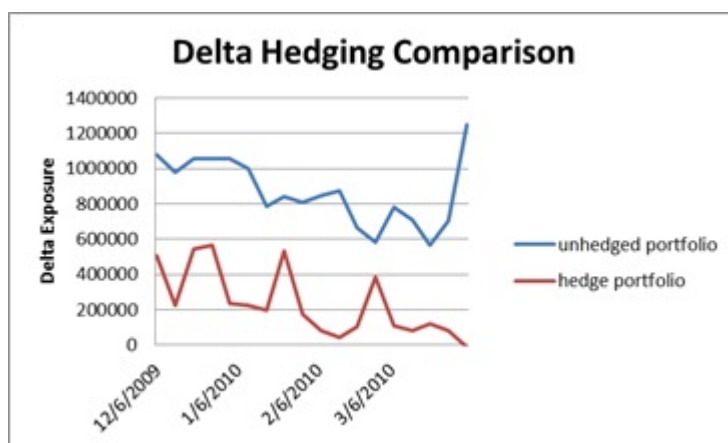


Figure 6.1: Delta Position Comparison

A variation of recursive partitioning is used on all subsequent results. The major contribution of this research is that it demonstrates that action-based features are a necessary component of trading strategy recovery. This research also illustrates that the temporal component is necessary to recover the decision space of a trading algorithm. To obtain the following results. Recursive partitioning was run on the first two months of trading data. Using the resulting partitions, the following two months of trading data were used as an out-of-sample test. The accuracy measures below indicate the percentage of possible actions correctly labeled in out-of-sample testing.

<i>AssetSymbol</i>	<i>StateFeatures</i>	<i>Action + StateFeatures</i>	<i>TemporalRelation</i>
AAPL	100%	99%	99%
AIG	100%	100%	100%
C	100%	84%	100%
DELL	98%	99.5%	100%
DIS	57%	91%	100%
GE	96.9%	96.9%	100%
GOLD	58%	100%	100%
GOOG	57%	100%	100%
KO	100%	100%	100%
MSFT	100%	82%	100%
RIG	94%	81%	100%
UBS	57%	89%	100%
Average	84.8%	93.5%	99.9

Table 6.1: Action Identification Accuracy

As can be seen in the above table, using only state space features results in the lowest accuracy rate. Although the state space plays a role in the algorithm's decision making, it is impossible to generate an accurate replica using the state space alone. This suggests that the algorithm is comparing possible actions. In other words, two actions lead to similar states, so the algorithm chooses the cheaper of the two options. Intuitively, however, we might expect the algorithm to make such a choice, since machine learning does take into account an action feature space.

Using action features in recursive partitioning greatly improves the strategy re-

covery. This approach might seem at first to be of limited value, because all the data gathered from it would appear to be presented in the form of unsorted and, more importantly, unrelated observations. However, this is not the case; there is in fact a temporal relationship among the observations. In other words, for any given day, when one possibility is chosen all other options can no longer be selected. Traditional recursive partitioning does not take this factor into account: even on days in which one option is selected, traditional recursive partitioning can select multiple options to trade.

This misclassification occurs because the partitions are not organized in the correct hierarchy. For example, if two options are currently held in inventory, traditional recursive partitioning could select to trade both options. However, this does not address the question of which option the algorithm determines to trade first. Therefore, it is important to partition based on correctly assigning all possible trades a positive or negative score for each day. In the study, the first recursive partition was used to determine how many trades would occur in a single day. Days in which two trades occurred were treated as two separate days: the first day was defined as containing all possible traded options; the second day contained all options minus the option that was just traded. The metric for splitting was designed to maximize the number of days correctly classified. A day was marked as correctly classified only if all trades were correctly labeled as traded or not traded. Once this was done, the following classification tree could be developed:

The above classification tree represents the strategy the team used to determine which options to trade. This strategy takes into account current states, the future state, and action comparisons. This relatively simple algorithm contained complicated

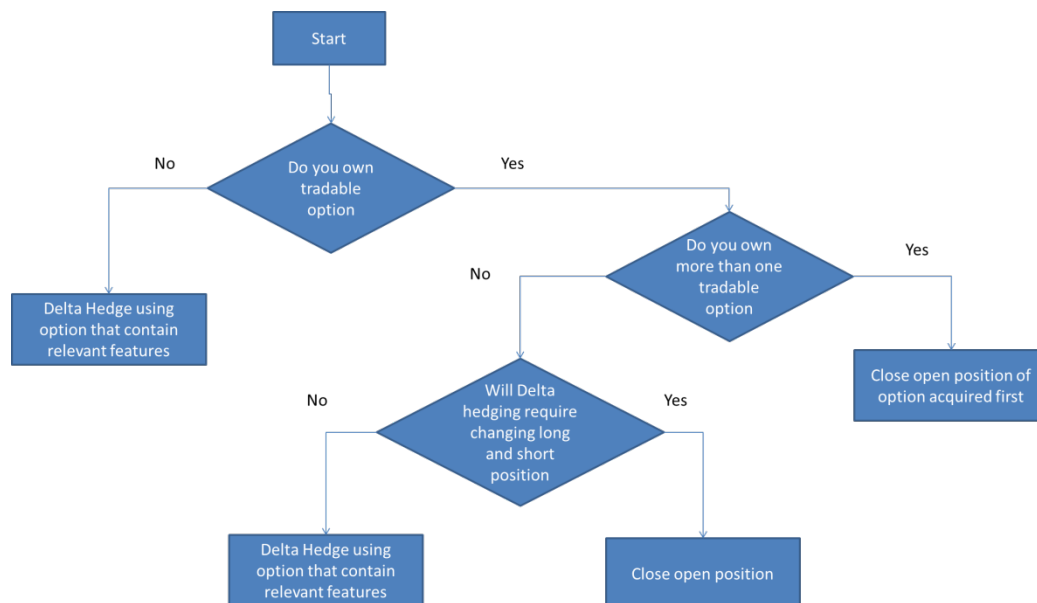


Figure 6.2: Security Selection Strategy

questions: Under what circumstances are days when two options in the same security are traded compared to days when only one is traded? Additionally, given that two options are traded, what option is traded first? To the author's knowledge, this is the first time a trade-level strategy such as the one outlined above has been recovered.

6.3 Results Determining Delta Exposure

It is important to recall that by determining the desired end-of-day delta exposure both the direction of a trade and the number of shares traded can be calculated. The same methodology reviewed in the previous section was used to examine the ability of regression trees to predict delta exposure. Each security family was treated as its own independent machine-learning problem. Regression trees were trained on the first two months of trade data for each security family, and the following two months

were used as out-of-sample validation results. Both the mean value prediction and the stepwise linear regression prediction were calculated for each day. The graph below shows that using stepwise linear regression can provide a better fit to the training data. Both variations of the regression tree are presented with the entire two months of observed data, which are used to create the classification partitions and generate predictions.

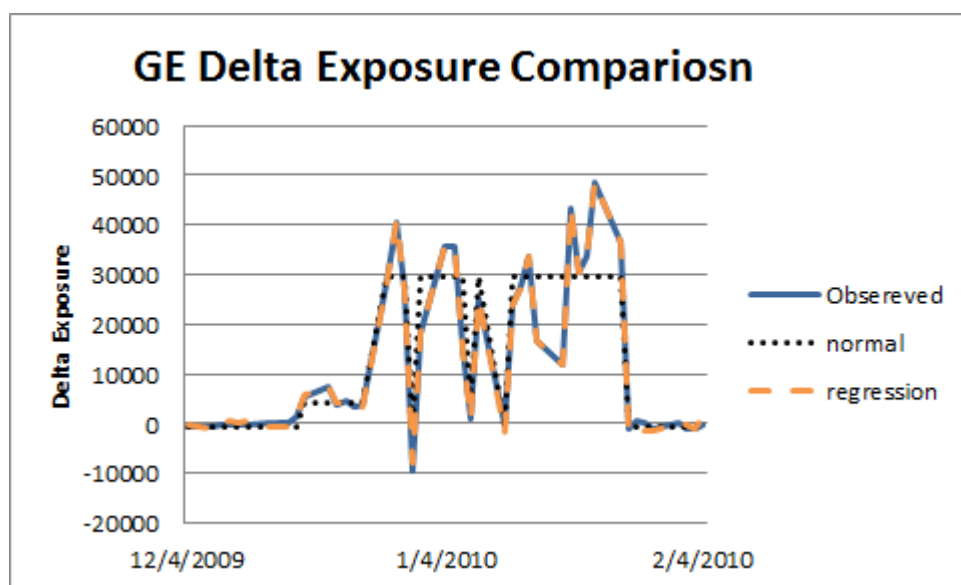


Figure 6.3: Training Set Delta Exposure Comparisons

The regression tree for GE is illustrated below. The first delta exposures given in the final nodes are the predictions using the mean of the dependent variables. The second delta exposures given are the predictions using stepwise linear regression.

Regression trees do not examine how an error in a prediction will affect subsequent decisions. Therefore, we implemented both regression tree outcomes and ran them over the entire four months. The results from the security family DELL best illustrate the overall finding.

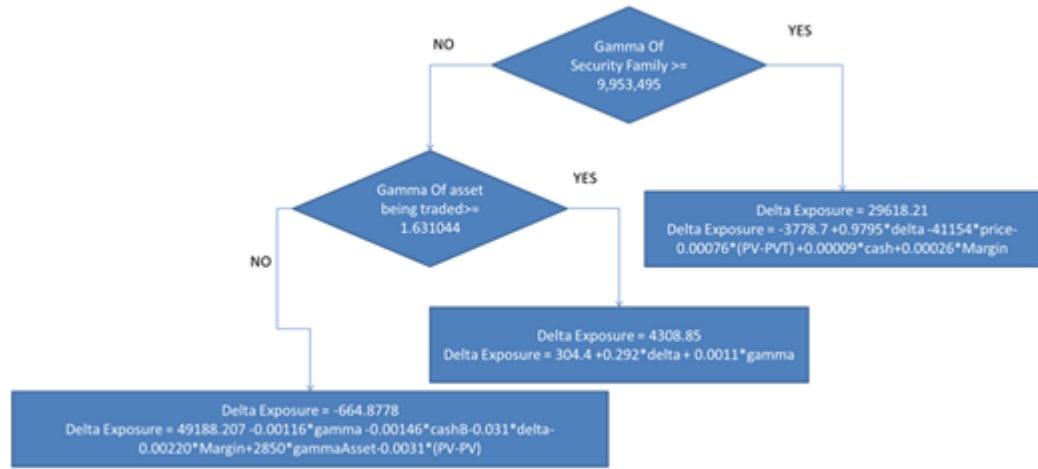


Figure 6.4: GE's Hedging Strategy

The above graph illustrates the observed delta exposure, the delta exposure recovered using the normal regression tree method, and the delta exposure recovered using the regression tree method augmented with stepwise linear regression. The red line indicates the transition from the training set to the validation set. The stepwise technique does a better job in matching the observed data in the training set. This trend continues into the validation set, until the regression technique makes a rapid and sudden divergence from the observed delta exposure. This occurs because the values presented to the linear equation are outside the training set values. Thus, the regression model generates an unexpected result. This is quickly corrected, but it illustrates an inherent instability in this type of methodology.

The regression tree method that uses the mean of the dependent variables as its prediction is slower at detecting state changes. This is due to the model using the average as the prediction of delta exposure. Since a large number of factors that determine desired delta exposure are intrinsic to the algorithm, such as cash on

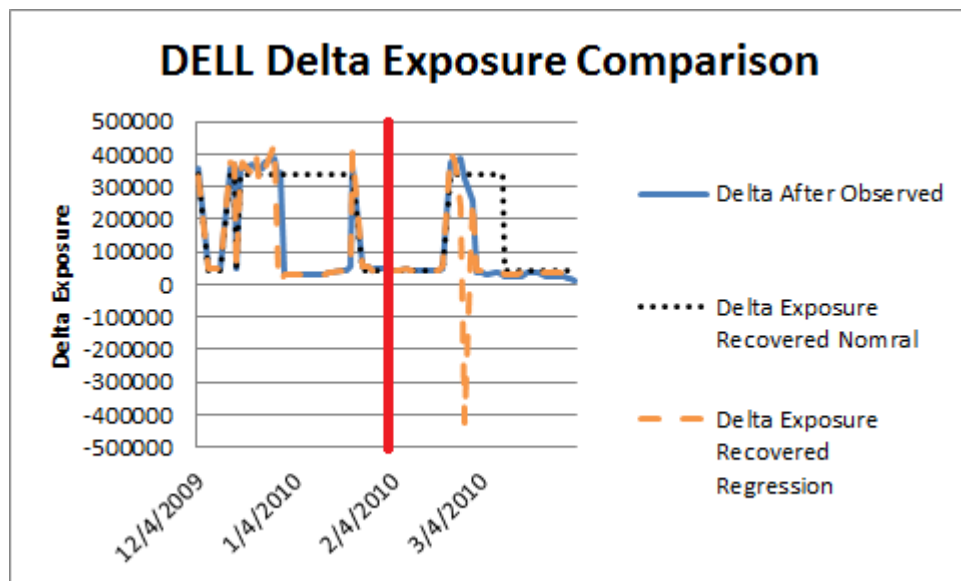


Figure 6.5: DELL'S Full Tournament Delta Exposure Comparison

hand, using the mean as the prediction causes these variables to change at a slower rate. Therefore, it takes longer for the strategy to recognize the difference in states. However, it does not have the sudden and rapid departure from the observed delta exposure that was seen in the regression model. This means the model is more stable than the stepwise regression model but not as responsive.

As stated above, the desired delta exposure is estimated so the direction and quantity of the trade can be predicted. Having this information allows us to examine how closely we match the observed algorithm. To determine which regression tree variation is better, the tracking error for the observed data is compared to both variations. Unlike the observed algorithm that is trying to minimize the tracking error, our recovered algorithm's goal is to match the observed algorithm tracking error. Below is a graph that illustrates how both the normal and stepwise regression methods compare with the observed algorithm:

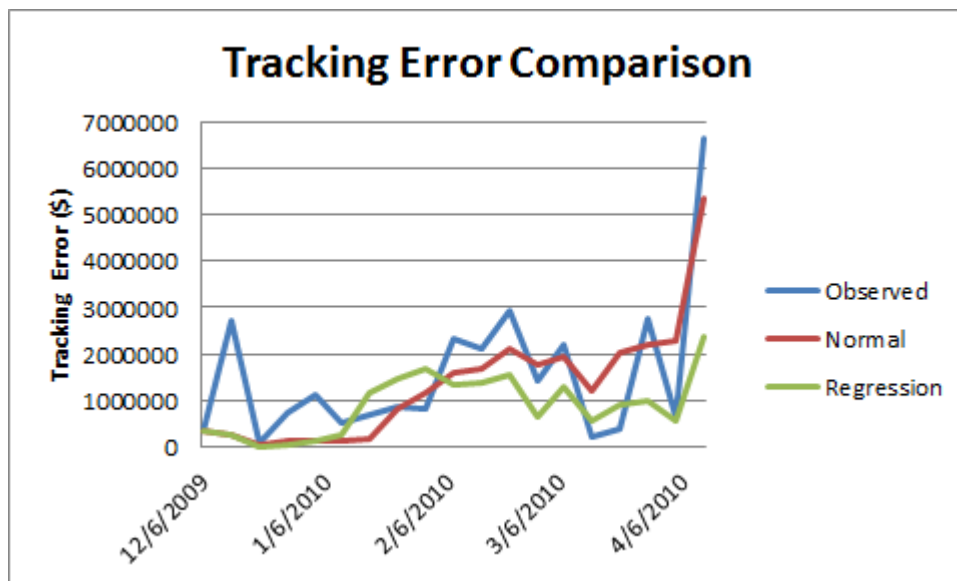


Figure 6.6: Tracking Error Comparison

This graph illustrates that both algorithms perform a better job of minimizing tracking error. However, the normal mean prediction regression tree does a better job of matching the observed algorithm. This is again due to the stability of the algorithm. Although it cannot predict the desired end-of-day delta exposure as well as the regression model, it does not suffer from erratic behavior. A cumulative difference between the observed algorithm tracking error and both variations of the regression trees tracking error is calculated and presented in the graph below. The ideal recovered algorithm would have a horizontal line at 0, meaning it exactly matched the tracking error of the observed algorithm. The graph shows that the mean prediction regression tree performs more accurately than the stepwise variation. This is because the mean prediction is stable and does not suffer as badly from values in data that have not been seen before.

The final section will offer a discussion of future work and lessons learned from

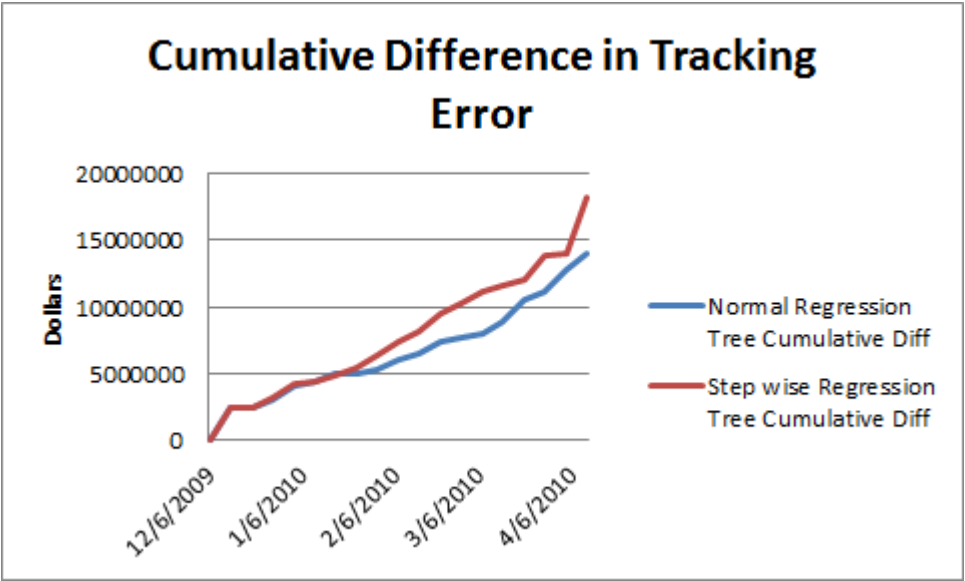


Figure 6.7: Cumulative Tracking Error Difference Between Observed and Recovered Algorithms

this research.

Chapter 7

Conclusions

The primary goal of this thesis was to prove that it is possible to recover the decision space of a trading strategy from trade-level data. To accomplish this goal, several unorthodox techniques had to be used. It was necessary to creating an action feature space to improve the accuracy of the recovered decision space. Treating actions as if they have properties made intuitive sense, although this has rarely been done in practice. And as this study demonstrated, generating an action feature space made it possible to use recursive partitioning directly in the recovery process.

The use of recursive partitioning illuminated the temporal importance of the data. Traditionally, machine learning treats observations as unrelated. However, the nature of the trading algorithm dictated that the temporal relation remain intact. Therefore, we extended the traditional recursive partitioning to maintain the relationship between observations. Although in this case the newly observed factor involved time, the parameters of recursive partitioning can be extended to include any type of relationship that must be maintained.

Two variations of regression trees were used to determine the desired delta exposure. It was determined that although stepwise regression can be used to augment a

regression tree's prediction, it has the tendency to overfit. Specifically, if out-of-sample data is provided to the recovered algorithm there is a potential for unpredictable behavior. Therefore, it is recommended that recovered algorithms be trained using the standard regression tree model. This model uses the mean of dependent variables that are categorized in the same node as the prediction for that class. This leads to a lag in recognizing state transitions. However, it prevents the aforementioned unexpected behaviors.

The results also illustrate that recursive partitioning and regression trees were unable to determine the decision space, which in the case of this study involved one state containing only one action. It was possible, however, to recover the methodology for determining which security would be traded, which should provide valuable insight into understanding how the trading algorithm works. However, perfectly determining the delta exposure was not possible. This could mean three things: First, it could indicate that the trading algorithm uses information outside the state space that was defined. Second, the algorithm could have partitions that are nonlinear and thus would not be found in the linear recursive partition and regression trees used in this paper. Third, since the delta of any particular option is based on an estimate of the volatility of the underlying stock, the volatility estimate used in this study could have differed from that of the trading competition participants.

This type of technique can help regulators better understand modern-day trading algorithms. Recovering trading algorithms will allow regulatory agencies to enforce existing policies; in particular, it will enable them to identify cases where traders have manipulated the market through misuse of algorithms. Furthermore, it will allow regulators to create artificial environments in which to test regulatory policies.

These environments can contain real-world trading algorithms, which will permit these agencies to conduct in-depth cost-benefit analyses prior to implementing new regulations. Such an artificial environment would also allow regulators to identify scenarios in which the market might move in an unexpected way.

Regulators are not the only party that might benefit from this type of technique. Exchanges could create the same type of artificial environment as described above. Within this environment, pricing incentives could be explored with the objective of making the exchange more profitable. Furthermore, the exchange could potentially sell artificial scenarios to private investors interested in training algorithms, an arrangement that would be mutually beneficial both to the exchange and to private investors. The more testing a private investor can do prior to launching an algorithm, the less likely the algorithm is to malfunction and cause a larger problem. Furthermore, by testing the algorithm over a wider range of scenarios, an investor can more effectively determine and optimize its profitability.

This thesis has illustrated some of the challenges associated with recovering strategies. In the artificial environment of the hedge fund tournament, it was possible to determine which asset would be traded. However, the study required employing some unorthodox techniques and extending powerful classification tools. It is concluded that it is feasible to recover trading strategies from detailed trade-level data.

Chapter 8

Appendix

8.1 State Space and Action Space Features

This section will list all features for both the State Space and Action Space. Additionally, a description for each feature is provided.

8.1.1 State Space Features - Security Selection Prediction

Below is a list of all the state space features, along with a description.

1. Day - Day of the week.
2. Stock Value - Underlying Stock Value.
3. Current Cash Account - the current cash in the account.
4. Current Margin Value - the amount of Margin open.
5. List of Open Positions - indicator value for each security. The value is 1 if there is an open position in the security and 0 otherwise.
6. Order of Acquisition - list the orders in which positions were open in securities.

7. List of shares owned in each security.

8.1.2 Action Space Features - Security Selection Prediction

This section list all the action space features, along with a description. All action space features are indicator values 1 or 0.

1. Security is an option - action will trade an option.
2. Security expires in May - action will trade an option that expires in May.
3. Security is already own - action will trade a security that has an open position.
4. Lowest price option - action will trade the lowest price (cheapest) option.
5. Lowest absolute value trade - action will result in the lowest absolute value for a trade. Assuming that all trades reduce delta exposure to zero and transaction cost is taken into account.
6. Largest delta/price ratio - action will trade option with the highest delta/price ratio.
7. Largest Short - generates most cash from shorting.
8. Sign Switch - action results in a switch from a long to short position or vice versa.

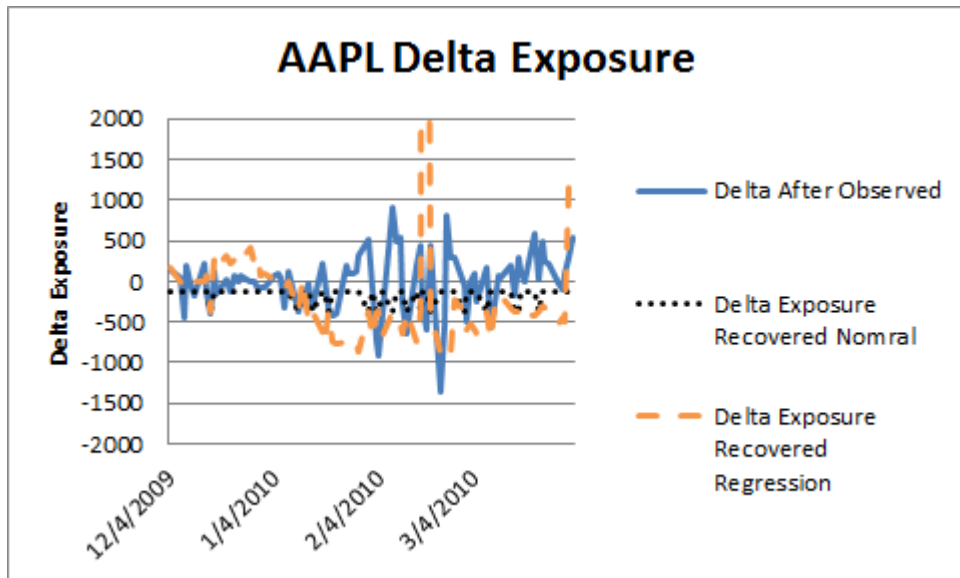
8.1.3 State Space Features - Deleta Exposure Prediction

This section list all the state space features used in delta exposure prediction

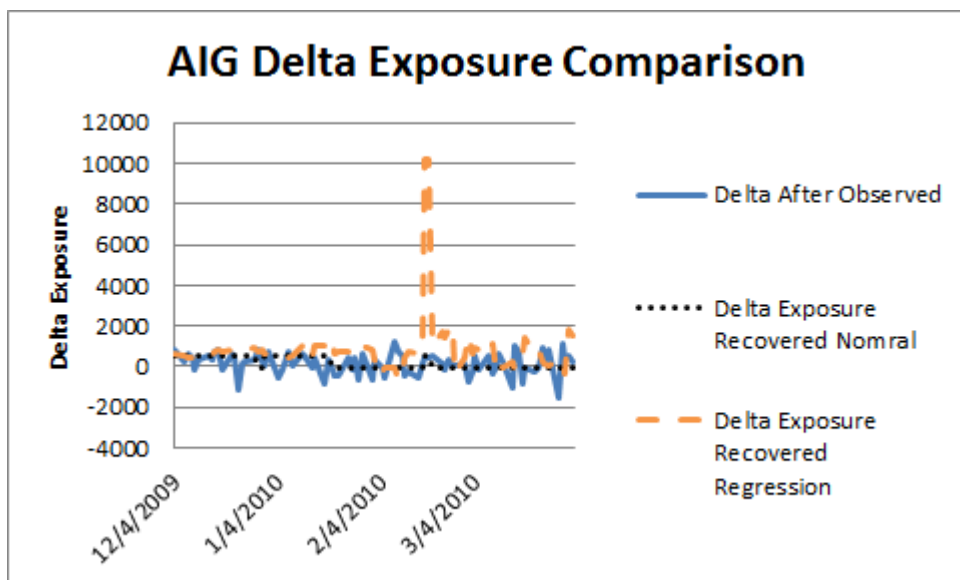
1. Security Family Delta Exposure: How much the value of securities currently held in inventory changes if the underlying stock value increases by \$1.
2. Security Family Gamma Exposure: How much the delta of securities currently held in inventory changes if the underlying stock value increases by \$1.
3. Delta of the Option being Traded.
4. Gamma of the Option being Traded.
5. Price of the Option being Traded.
6. Current Cash Account: the current cash in the account.
7. Current Margin Value: the amount of Margin open.
8. Portfolio Value - Target Value : difference between the current portfolio value and the target value.

8.2 Delta Exposure Comparison

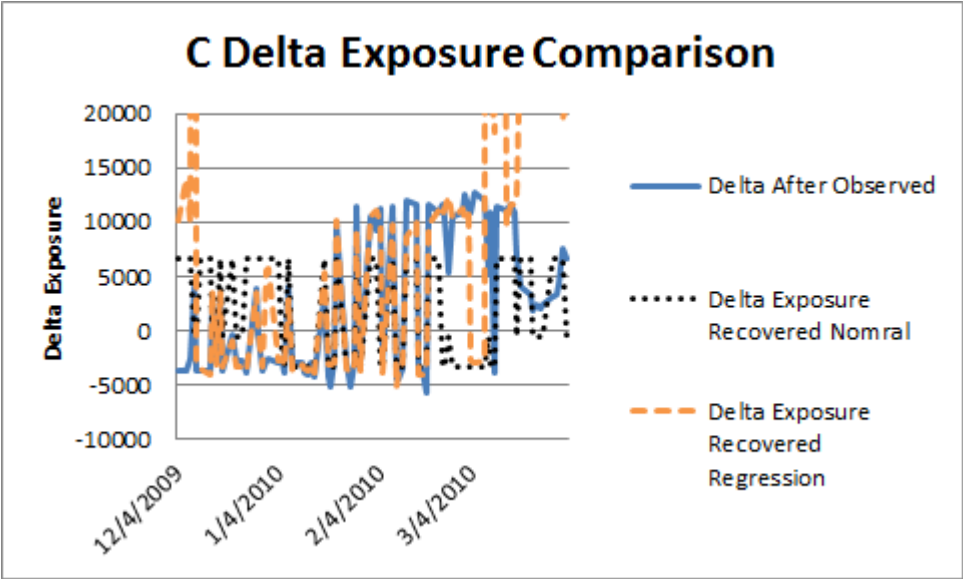
8.2.1 AAPL



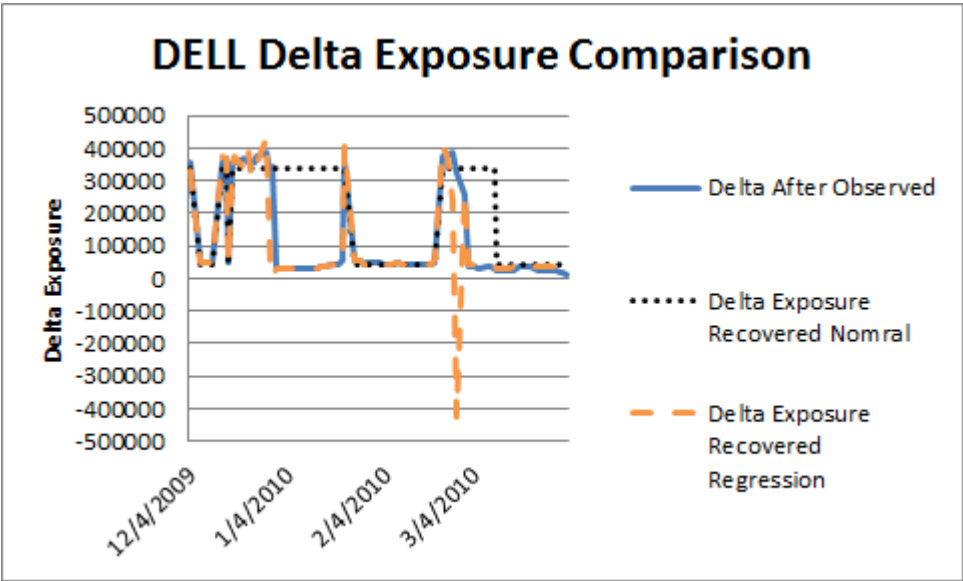
8.2.2 AIG



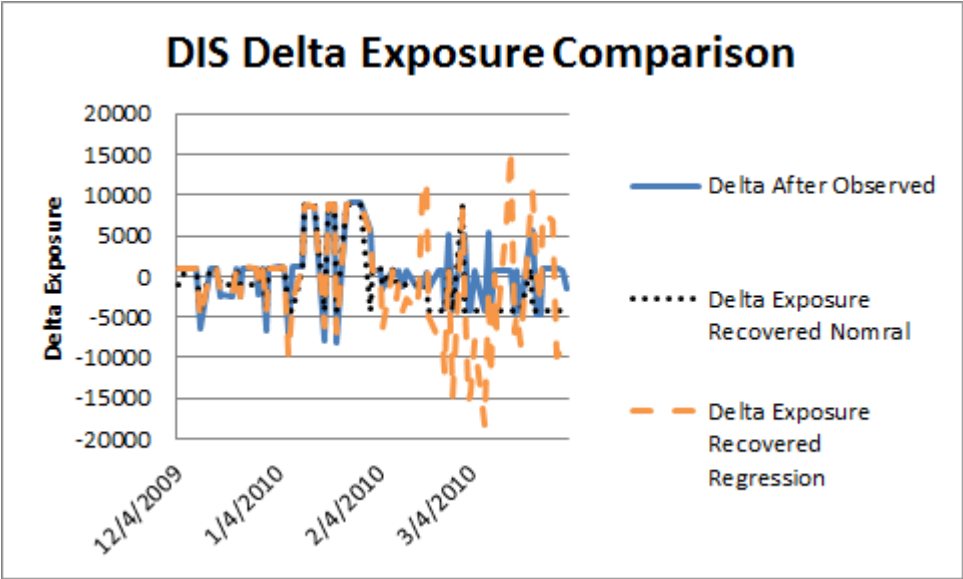
8.2.3 C



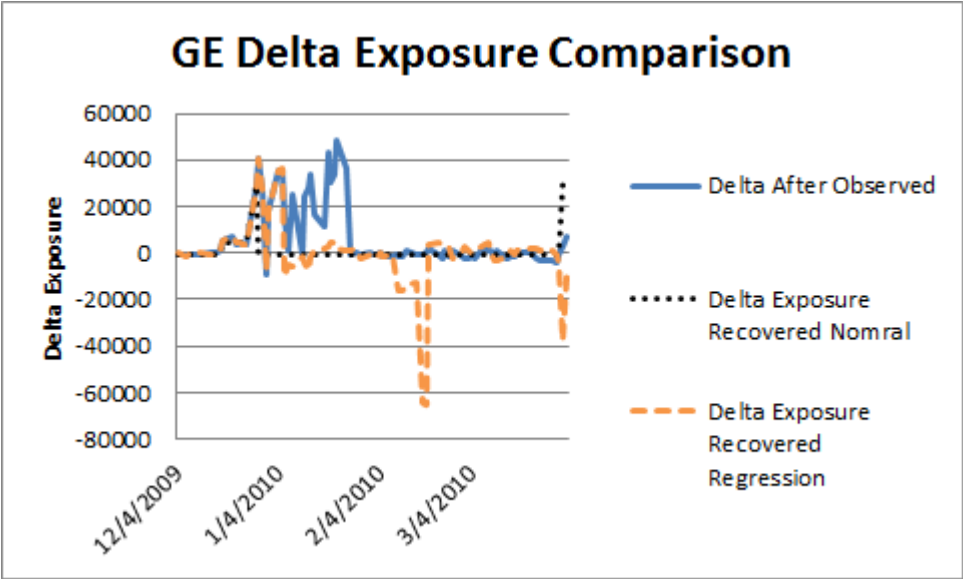
8.2.4 DELL



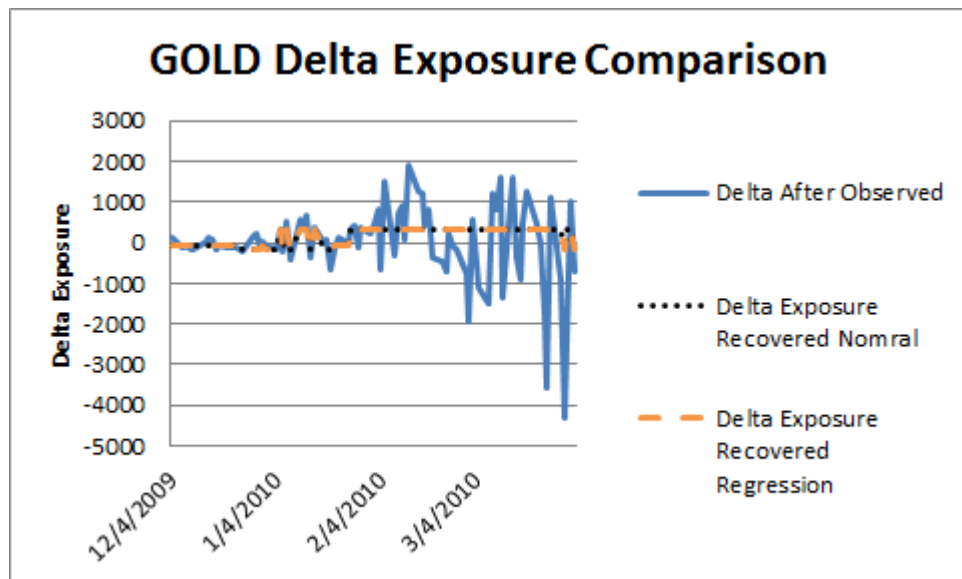
8.2.5 DIS



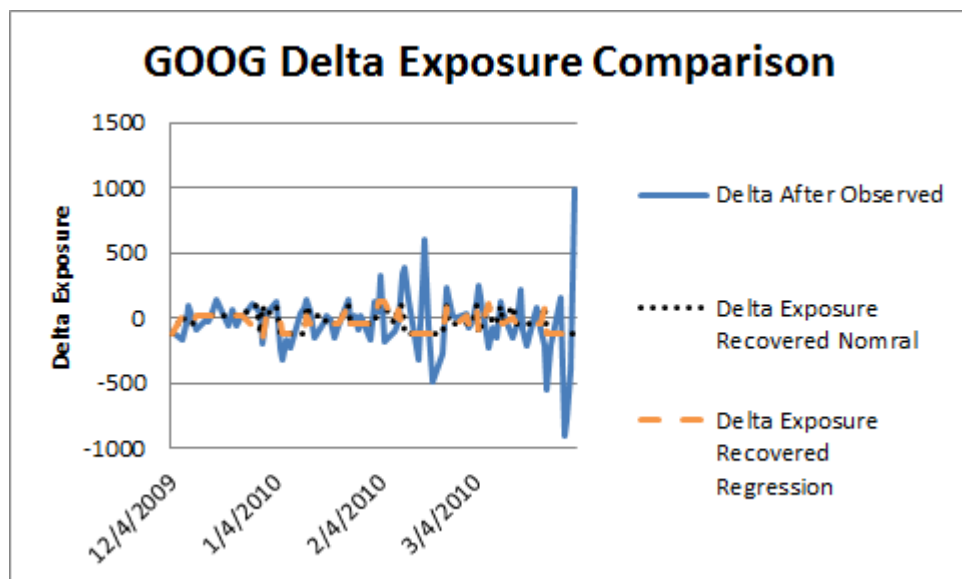
8.2.6 GE



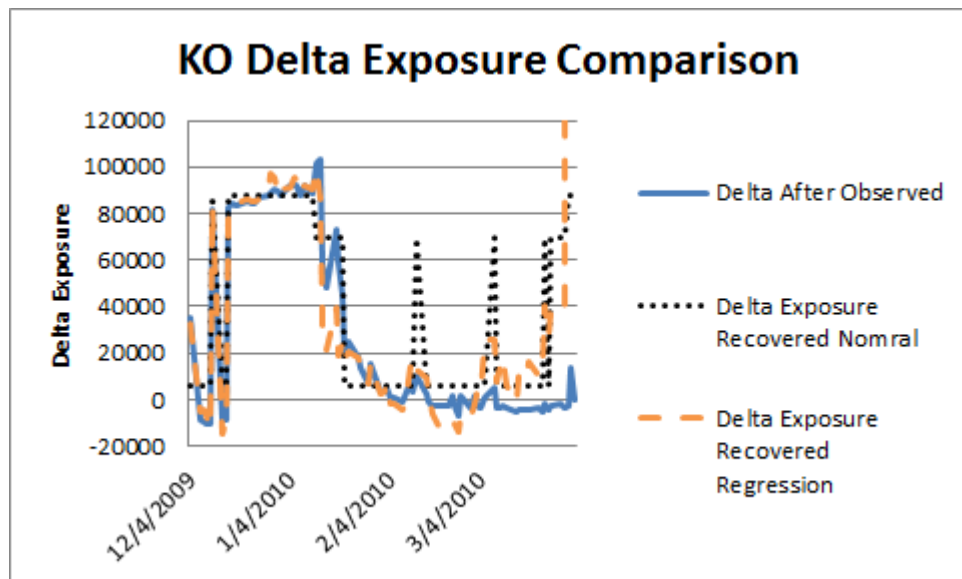
8.2.7 GOLD



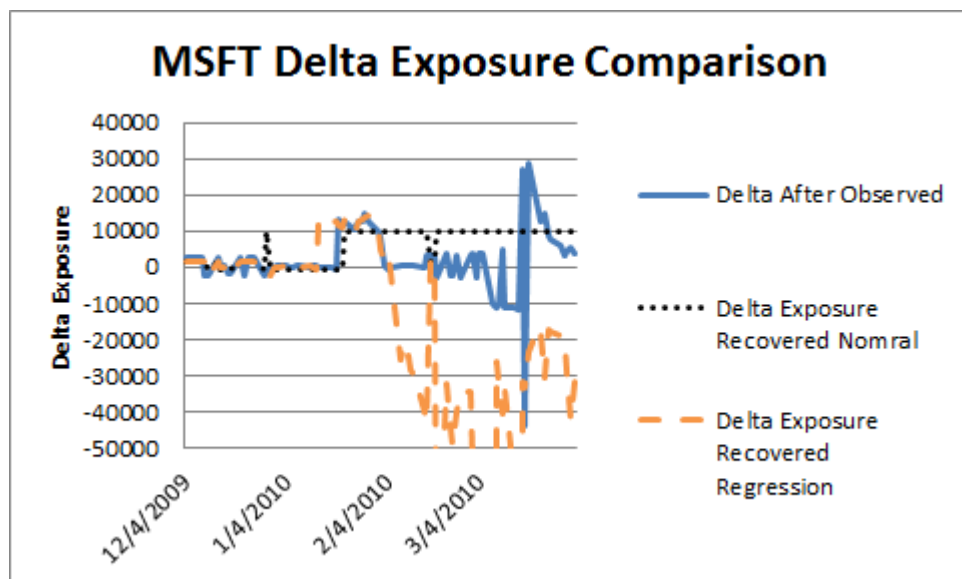
8.2.8 GOOG



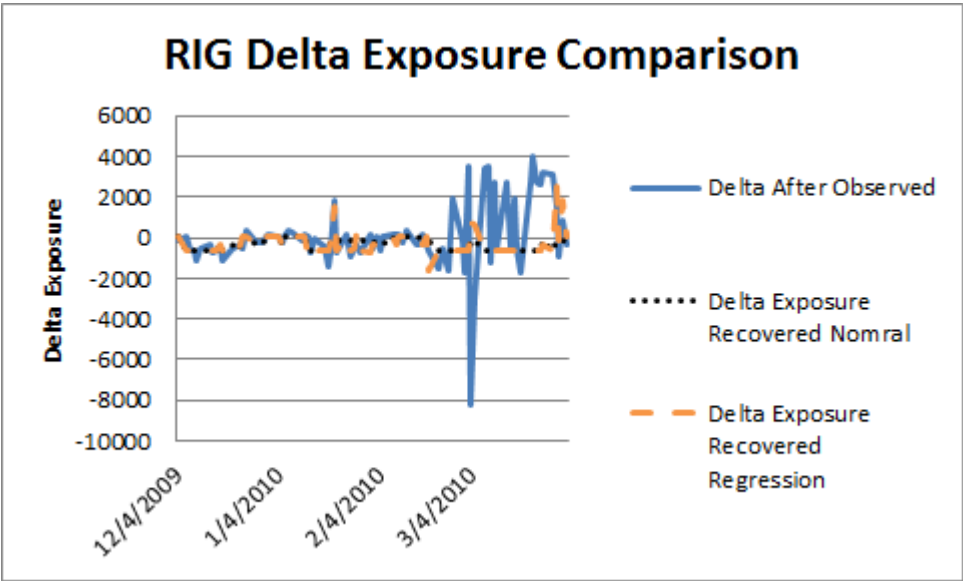
8.2.9 KO



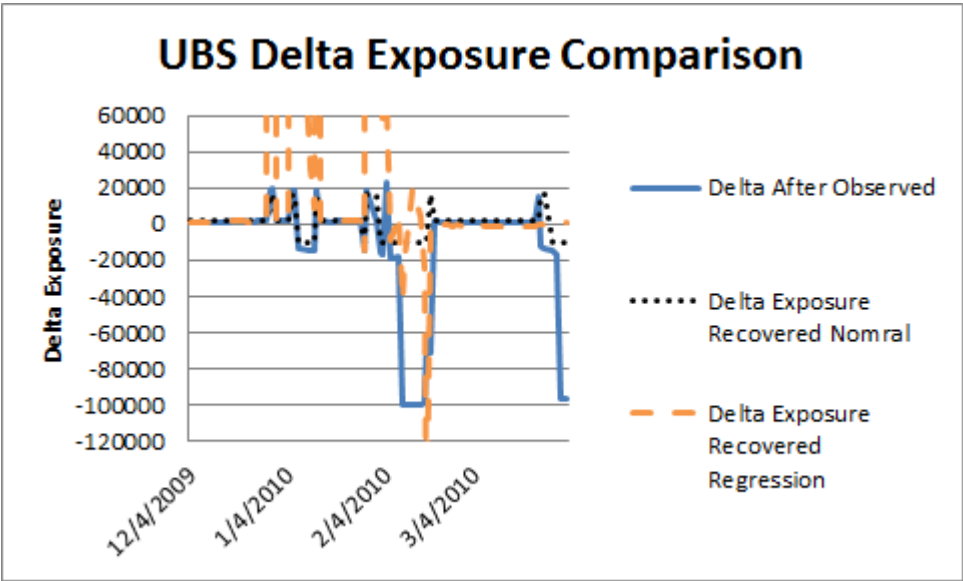
8.2.10 MSFT



8.2.11 RIG



8.2.12 UBS



Bibliography

- [1] S.H. Chen and C. H. Yeh, “On the emergent properties of artificial stock markets: The efficient market hypothesis and the rational expectations hypothesis,” *Journal of Economic Behavior and Organization*, vol. 49, pp. 217–239, 2002.
- [2] R. Hayes, M. Paddrik, A. Todd, S. Yang, P. Beling, and W. Scherer, “Agent based model of the e-mini s&p 500 future: Application for policy making,” in *Winter Simulation Conference*, 2012.
- [3] A. Todd S. Yang P. Beling M. Paddrik, R. Hayes and W. Scherer, “An agent based model of the e-mini s&p 500 and the flash crash,” in *Computational Intelligence for Financial Engineering and Economics*, 2011.
- [4] S. Mike and J. Farmer, “An empirical behavioral model of liquidity and volatility,” *Journal of Economics Dynamics and Control*, vol. 32, pp. 200–234, 2008.
- [5] S. Morris and H. Shin, “Financial regulation in a system context,” *Brookings Papers on Economic Activity*, vol. Fall, pp. 229–274, 2008.
- [6] J. Holland and J. Miller, “Artificial adaptive agents in economic theory,” *The American Economic Review*, vol. 81, pp. 365–370, 1991.

- [7] B. Lebaron, “Building the santa fe artificial stock market. working paper, graduate,” in *School of International Economics and Finance, Brandeis*, 2002, pp. 1117–1147.
- [8] S. Maslov, “Simple model of a limit order-driven market,” *Physica A: Statistical Mechanics and its Applications*, vol. 278, pp. 571–578, 2002.
- [9] J. Conlisk, “Why bounded rationality,” *Journal of Economic Literature*, vol. June, pp. 669–700, 1996.
- [10] R. Shiller, “Human behavior and the efficiency of the financial system,” *Handbook of macroeconomics*, pp. 1305–1340, 1999.
- [11] H. Simon, “Theories of decision-making in economics and behavioral science,” *The American Economic Review*, vol. 49, pp. 253–283, 1959.
- [12] P. Abbeel and A. NG, “Apprenticeship learning via inverse reinforcement learning,” in *21st International Conference on Machine Learning*, 2004.
- [13] A. Ng and S. Russel, “Algorithms for inverse reinforcement learning,” in *International Conference on Machine Learning*, 2000.
- [14] R. Hayes A. Todd P. Beling S. Yang, M. Paddrik and W. Scherer, “Behavior based learning in identifying high frequency trading strategies,” in *Computational Intelligence for Financial Engineering and Economics*, 2011.
- [15] CFTC and SEC, “Findings regarding the market events of may 6, 2010,” Tech. Rep.

- [16] M. Samadi A. Kirilenko, A. S. Kyle and T. Tuzun, “The flash crash: The impact of high frequency trading on an electronic market,” Tech. Rep., Commodity Future Trade Commission, 2011.
- [17] W. Sharpe, “Asset allocation,” *The Journal of Portfolio Management*, vol. 18, pp. 7–19, 1992.
- [18] W. Fung and D. Hsieh, “Empirical characteristics of dynamic trading strategies: The case of hedge funds,” *The Review of Financial Studies*, vol. 10, pp. 275–302, 1997.
- [19] S. Brown and W. Goetzmann, “Mutual fund styles,” *Journal of Financial Economics*, vol. 43, pp. 373–399, 1997.
- [20] B. Liang, “On the performance of hedge funds,” Tech. Rep., University of Massachusetts at Amherst, 1998.
- [21] J. Hasanhodzic and A. Lo, “Can hedge-fund returns be replicated?: The linear case,” *The Linear Case (August 16, 2006)*, 2006.
- [22] J. Meyfredi N. Amenc, L. Martellini and V. Ziemann, “Passive hedge fund replication—beyond the linear case,” *European Financial Management*, vol. 16, pp. 191–210, 2010.
- [23] R. McDonald, *Derivative Markets*, vol. 2, Pearson Education, Boston, MA, June 2006.
- [24] L. Kogan D. Bertsimas and A. Lo, “Hedging derivative securities and incomplete markets: An-arbitrage approach,” *Operations Research*, vol. 49, pp. 372–397, 2001.

- [25] H. Kat and H. Palaro, “Who needs hedge funds? a copula-based approach to hedge fund return replication,” *A Copula-Based Approach to Hedge Fund Return Replication (November 23, 2005). Alternative Investment Research Centre Working Paper*, , no. 27, 2005.
- [26] P. beling Q. Qiao, “Inverse reinforcement learning with gaussian process,” in *American Control Conference*, 2011.
- [27] R. Bellman, *Dynamic Programing*, vol. 1, Princeton Univ Press, 1957.
- [28] U. Syed and R. Schapire, “A game-theoretic approach to apprenticeship learning,” *Advances in neural information processing systems*, vol. 20, pp. 1–8, 2008.
- [29] G. Neu and C. Szepesvári, “Apprenticeship learning using inverse reinforcement learning and gradient methods,” *arXiv preprint arXiv:1206.5264*, 2012.
- [30] K. Dvijotham and E. Todorov, “Inverse optimal control with linearly-solvable mdps,” in *Proceedings of the Interntional Conference on Machine Learning. Citeseer*, 2010.
- [31] M. Quigley P. Abbeel, A. Coates and A. Ng, “An application of reinforcement learning to aerobatic helicopter flight,” *Advances in neural information processing systems*, vol. 19, pp. 1, 2007.
- [32] B.D. Ziebart, A. Maas, J.A. Bagnell, and A.K. Dey, “Maximum entropy inverse reinforcement learning,” in *Proc. AAAI*, 2008, pp. 1433–1438.
- [33] P. Beling W. Scherer A. Kirilenko S. Yang, Q. Qiao, “Gaussian process based trading strategy identification,” Tech. Rep., System and Information Engineering, University of Virginia.

- [34] T. Therneau and E. Atkinson, “An introduction to recursive partitioning using the rpart routines,” Tech. Rep., Technical Report 61, Section of Biostatistics, Mayo Clinic, Rochester., 1997.
- [35] M. Kutner, C. Nachtsheim, J. Neter, and W. Li, *Applied Linear Statistical Models*, vol. 5, McGraw-Hill/Irwin, 2005.