Agents in the Market Place
An Exploratory Study on Using Intelligent
Agents to Trade Financial Instruments

Rui Pedro da Costa Barbosa

UMinho | 2011

Universidade do Minho
Escola de Engenharia

Rui Pedro da Costa Barbosa

Agents in the Market Place
An Exploratory Study on Using Intelligent Agents to Trade Financial Instruments

Maio de 2011

Universidade do Minho
Escola de Engenharia

Rui Pedro da Costa Barbosa

Agents in the Market Place
An Exploratory Study on Using Intelligent Agents to Trade Financial Instruments

Tese de Doutoramento em Informática

Trabalho realizado sob a orientação do
Professor Doutor Orlando Belo

Maio de 2011

This work was sponsored by the FCT – Fundação para a Ciência e a Tecnologia – and the QREN – Quadro de Referência Estratégico Nacional.

Este trabalho foi financiado pela FCT – Fundação para a Ciência e a Tecnologia – e pelo QREN – Quadro de Referência Estratégico Nacional.

Thesis Advisor

Orlando Belo

Author

Rui Pedro Barbosa

Agents in the Market Place

Abstract

This dissertation documents our exploratory research aimed at investigating the utilization of intelligent agents in the development of automated financial trading strategies. In order to demonstrate this potential use for agent technology, we propose a hybrid cognitive architecture meant for the creation of autonomous agents capable of trading different types of financial instruments. This architecture was used to implement 10 currency trading agents and 25 stock trading agents. Their overall performance, evaluated according to the cumulative return and the maximum drawdown metrics, was found to be acceptable in a reasonably long simulation period. In order to improve this performance, we defined negotiation protocols that allowed the integration of the 35 trading agents in a multi-agent system, which proved to be better suited for withstanding sudden market events, due to the diversification of the investments. This system obtained very promising results, and remains open to many obvious improvements. Our findings lead us to conclude that there is indeed a place for intelligent agents in the financial industry; in particular, they hold the potential to be employed in the establishment of investment companies where software agents make all the trading decisions, with human intervention being relegated to simple administrative tasks.

Keywords: *Intelligent Agents, Multi-Agent Systems, Data Mining, Financial Trading.*

Orientador

Orlando Belo

Autor

Rui Pedro Barbosa

Agentes no Mercado

Resumo

Esta dissertação documenta um estudo exploratório destinado a investigar a utilização de agentes inteligentes no desenvolvimento de estratégias de investimento financeiro automatizadas. Para demonstrar este uso potencial para tecnologia de agentes, foi proposta uma arquitectura cognitiva híbrida destinada à criação de agentes autónomos capazes de negociar diferentes tipos de instrumentos financeiros. Esta arquitectura foi utilizada para implementar 10 agentes que negoceiam pares cambiais, e 25 agentes que negoceiam acções. A performance global destes agentes, avaliada de acordo com as métricas de retorno acumulado e *drawdown* máximo, foi considerada aceitável ao longo de um período de simulação relativamente longo. Para melhorar esta performance, foram definidos protocolos de negociação que permitiram a integração dos 35 agentes num sistema multi-agente, que demonstrou estar melhor preparado para enfrentar alterações súbitas nos mercados, devido à diversificação dos investimentos. Este sistema obteve resultados muito promissores, e pode ainda ser sujeito a diversos melhoramentos. Os nossos resultados indiciam que os agentes inteligentes podem ocupar um lugar de relevo na indústria financeira; em particular, aparentam ter potencial suficiente para serem aplicados na criação de fundos de investimento onde todas as decisões de negociação são efectuadas por agentes de *software*, sendo a intervenção humana relegada para tarefas administrativas básicas.

Palavras-chave: *Agentes Inteligentes, Sistemas Multi-Agente, Mineração de Dados, Negociação Financeira.*

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Trading in financial markets is undergoing a radical transformation, one in which algorithmic methods are becoming increasingly more important. This transformation is the result of the "technological arms race" (Hasanhodzic *et al.*, 2009) being carried out by numerous quantitative trading firms, in their never ending quest for an edge over competitors. Algorithmic trading, a form of financial trading in which computer programs are put in charge of opening and closing trades without human intervention, is quickly becoming the norm in many markets. In their search for better algorithms, several investment companies have been experimenting with *Artificial Intelligence* (AI) methods, and some are now advertising their complete reliance on AI-based trading strategies. However, except for a few buzzwords, little is known about the actual implementation of these strategies, or the trading results that may be expected from them. Thus, it is currently very hard to tell if there is any substance to all the hype surrounding the application of artificial intelligence in financial trading. The work that will be described in this thesis intends to shed some light on this issue: we will be presenting an innovative way to utilize artificial intelligence techniques in the development of automated trading strategies, and will evaluate these strategies for safety and profitability. More specifically, we will describe and test a method for creating intelligent agents that can trade financial instruments autonomously. These agents are meant to be the software equivalent of human traders – they are the next logical step forward in the aforementioned "arms race". Since this is an interdisciplinary study, relating to the fields of financial trading, agency and artificial intelligence, we will begin with a brief introduction to these fields, and present some fundamental concepts that are needed to fully understand the work that will be discussed later.

1.1 Basic Concepts in Financial Trading

In the world of finance, a trade is defined as a transaction involving a financial instrument, usually with the expectation of a positive return. With the exception of arbitrage and hedged trades, most of these transactions have considerable risk associated. That is to say, financial trading is an inherently dangerous activity. With the advent of online brokers, it is now easier than ever for retail traders to engage in the speculative trading of many different types of financial instruments, among which stocks, commodities, bonds and currency pairs. Each type is traded in a specific market with unique operating characteristics. In the stock market, for example, traders can buy or sell stocks of publicly traded companies. If the price of a stock is expected to increase, the trader buys the stock, hoping to sell it for a higher price at a later time. This action, commonly referred to as "going long", is the best known form of trading. If the stock is sold at a higher price, the trader makes money; if it is sold at a lower price, the trader loses money. Traders can also profit from falling prices, by short selling stocks whose prices are expected to decline. This is known as "going short", and is accomplished by borrowing the stock from a third party, and selling it in the market. If the price of the stock drops, the trader will buy back the stock at the lower price, return it to the lender, and make a profit in the process. However, if the stock buyback – known as short covering – occurs at a higher price, the trader will lose money.

When buying a company's stock, the worst-case scenario for a long trader is the company going bankrupt, and the stock price going to zero. A regular cash account can be used for this type of trading, which requires the trader to pay the full amount when the stock is bought. If the price drops to zero, the trader “only” loses the money invested in the stock. On the other hand, when shorting a stock, there is no limit to the potential loss, because there is no upper bound for the stock’s price. For this reason, short selling requires a margin account; when trading on margin, the broker lends funds to the trader, and the cash and securities in the account are utilized as collateral for the loan. The trader will be borrowing funds whenever the total amount invested surpasses the collateral; when this happens, the trader is said to be using leverage, which implies that both the investment gains and the losses will be magnified. The maximum leverage allowed varies from broker to broker; if, for example, the broker permits a maximum leverage of 4:1, that means the trader is allowed to invest up to 4 times the collateral available. Leveraged trades may result in a loss bigger than the collateral, so the trader runs the risk of losing all the money in the account, and still owing money to the broker. In order to protect themselves, brokers require traders to keep enough collateral at all times. If this requirement is not met, the trader will be warned to either close some of the trades, or to deposit more money in the account. This warning is known as a margin call. If the trader is unable or unwilling to rectify the situation, the broker will forcefully close the trades. But even with this prevention mechanism in place, a trader may still experience losses that surpass the capital in the trading account; for instance, suppose the trader has \$1,000, and decides to buy a company’s stock using 4:1 leverage, i.e., a \$4,000 investment. Now imagine this company declares bankruptcy right after the market closes. The stock’s opening price in the next trading day will surely be close to zero, which means the trader will get an automatic margin call as soon as the market opens; thus, the trader will lose the \$1,000 collateral in the account, and will owe up to \$3,000 to the broker. Conversely, if things go the trader’s way, leverage will be extremely beneficial. Imagine that, instead of going bankrupt, the company announced that it was about to be bought out by another company, and that this announcement made the price of the stock double overnight. If the trader was not using leverage, the return obtained would have been “only” 100%. But since 4:1 leverage was used, the return on the investment will be 4 times greater, or 400%. This example goes to show that leverage is a double-edged sword: it can greatly increase profits, but may also lead to disastrous results.

The concepts presented so far apply not only to the stock market, but also to other financial markets, like the foreign exchange (or Forex) market. The Forex market is the place where currency exchange rates are set, through the trading of currency pairs. As of late, it has been gaining increasingly more attention from retail traders. Unlike the stock market, the Forex market is not centralized, i.e., there is no central exchange to trade currencies. Only recently have retail traders been able to participate in this market directly, with the advent of electronic communication networks, or ECNs. An ECN gathers bid and ask prices from several liquidity providers, and allows traders to make their own bid and ask offers. Since currency pairs are traded in many different venues, there is no unique exchange rate for each pair; nevertheless, the prices quoted by different liquidity providers are usually very similar, because the market is efficient enough to eliminate these arbitrage opportunities. Another unique characteristic of the Forex market is its long trading hours: currencies can be traded nonstop during work days. A speculative investment in a currency pair is frequently based on the relative economic performances of the corresponding countries or unions. The pair's price represents the amount that is needed of the second currency (the quote currency) to buy one unit of the first currency (the base currency). If the trader predicts the base currency will become more valuable compared to the quote, it goes long the pair; in practical terms, this means the trader will buy the base currency and short sell the quote currency. If, on the other hand, the base currency is expected to become less valuable, the trader short sells the pair, which implies short selling the base and buying the quote. Take the USD/JPY pair, for example. This pair's price corresponds to the price of one United States dollar ($), the base currency, expressed in Japanese yen (¥), the quote currency. A price of 95.43 for this pair means that we need ¥95.43 to buy $1. If the dollar is expected to become more valuable compared to the yen, the trader should buy the pair, which would originate a long dollar exposure and a short yen exposure. If the expectation is for the yen to become more valuable, the trader should short sell the pair, which would result in a short dollar exposure and a long yen exposure. The movements in currency prices are usually measured in pips. A pip is the smallest possible change in the price of a currency pair; considering only the most frequently traded pairs, a pip corresponds to a price movement of 0.01 for those in which the Japanese yen is the quote currency, and a movement of 0.0001 for all the others.

Regardless of the market and the instrument being negotiated, traders always interact with the market using orders, of which there are many different types. The most commonly used are the market orders and the limit orders: a market order is a request to buy or sell a financial instrument at the current market price, while a limit order is a request to buy or sell the instrument at a specific price. Take-profit and stop-loss orders, believed to be an important part of successful trading strategies, are usually implemented using limit orders. A take-profit order is used to automatically close a trade if it reaches a predefined profit target, while a stop-loss order is used to close a trade if it reaches a specified maximum loss, to prevent that loss from widening.

The decision to send a buy or a sell order to any given market is traditionally based on one (or a combination) of three types of analysis: fundamental, technical or quantitative. Fundamental analysis entails studying the financial health of the entity underlying the financial instrument, as well as that of its competitors, in order to determine if the instrument is undervalued or overvalued. Technical analysis, usually applied to shorter time frames, consists in using the instrument's historical prices to forecast its price in the future; this is usually accomplished with the identification of support and resistance levels in the instrument's price chart, or by using price-based indicators such as the relative strength index (RSI), the Williams %R, moving averages, the moving average convergence/divergence (MACD), among many others. Finally, quantitative analysis implies using mathematical and statistical models to make financial predictions; this is the most complex type of analysis, and the most relevant to our work. There are numerous studies demonstrating how quantitative analysis can be performed using artificial intelligence techniques. In particular, several researchers have shown that data mining may be useful for carrying out this task, and some investment companies advertise their use of data mining models as part of their trading strategies. We will be taking this into consideration, once we start researching the design of autonomous trading agents.

No matter what type of analysis is employed to decide when to buy or short sell a financial instrument, the end result is always a speculative forecast for the instrument's price in the future. Despite the huge investment industry surrounding financial markets, this idea that instrument prices are predictable is far from consensual. Fama (1970) postulated in his famous efficient market hypothesis that, at any given point in time, an instrument's price always fully reflects all the information available. He distinguished between three different forms of market efficiency: the weak form of the efficient market hypothesis states that historical prices are of no use in predicting the instrument's price in the future; the semi-strong form makes the same claim, but adds that all publicly available information is also reflected in the instrument's price at all times, and therefore cannot be utilized to predict future prices; the strong form of the efficient market hypothesis is even more restrictive, stating that neither historical prices, nor publicly available or insider information allow for superior risk-adjusted returns. We can infer from Fama's hypothesis that forecasting an instrument's price direction should be an impossible task. As he puts it, there is no such thing as an undervalued or an overvalued asset, because an asset's market price always fully reflects all known information, including the traders' expectations regarding its performance in the future. Besides Fama's, there are several other hypotheses stating that financial prices cannot be predicted. The random walk hypothesis, popularized by Malkiel (1985) in his famous book "A Random Walk Down Wall Street", postulates that stock prices follow a random walk model, i.e., they evolve gradually as a sequence of random changes; this means that trying to predict stock prices is a silly endeavour, no matter how sophisticated the forecasting method. To prove his point, Malkiel conducted a very telling experiment. First, he used a random walk to generate a price chart for a stock: starting with a price of $50, he evolved the price by continuously adding or subtracting a random quantity from the previous price. The corresponding chart was shown to a technical analyst; unaware of how the prices were generated, the analyst promptly detected an upward trend in the stock's price, and suggested that the stock should be bought. Not only did this prove that the expert could not tell the difference between a random walk chart and a real price chart, it also demonstrated that, just because a technical analyst is able to find familiar patterns in a chart, that does not mean that those patterns have any sort of predictive power. The martingale hypothesis, advocated by Samuelson (1965), is another popular hypothesis that attempts to model the behaviour of stock prices. It is less restrictive than the random walk hypothesis: it also postulates that forecasting based on historical prices is ineffective, but adds that the best forecast for an instrument's future price is its current price. All in all, the same conclusion: there is no point in trying to forecast financial prices.

It would be easy to use anecdotal evidence to dismiss claims that the markets cannot be predicted. Several famous investors, like Warren Buffett and Jim Rogers, have been successful for decades. This means that, more often than not, they have been capable of making profitable financial forecasts, something that should not be possible if asset prices were completely random. However, it is possible that these success stories could be due to chance alone. Buffett (1984) puts it best in his analogy entitled "The Superinvestors of Graham-and-Doddsville". This story revolves around a coin flipping competition with 225 million participants, each betting one dollar. They are divided into groups of two, and in each group the person that correctly calls a coin flip gets the other person's dollar, and the loser leaves the competition. In the following day, the remaining competitors bet all their winnings in another round of coin flipping. After just 20 days, there will be a group of 215 people that were able to successfully call 20 coin flips in a row; these people will have turned their initial "investment" of $1 into a little over $1 million. If they were picking stocks instead of predicting coin flips, they would surely be praised for their amazing trading skills. However, as Buffett points out, we would get a similar group of "experts" if the competition was started with 225 million orangutans. The major implication of this analogy is that there is always the possibility that a trader's success is due to luck, rather than talent. Given the large number of entities participating in the markets, it is inevitable that a few statistical outliers will achieve consistent profitability just because they are lucky. This is not the point that Buffett intended to make with his story; he goes on to state that, even if some traders could owe their success to luck, most elite traders share the philosophy of value investing, which is based on fundamental analysis. He extrapolated from this empirical observation that that characteristic is what made them successful, i.e., their profitability is not due to chance, but rather to their ability to find stocks whose market prices are too low, compared to their intrinsic value. Unfortunately, there is no way to know if this correlation does in fact imply causation. Regardless, Buffett's position that financial markets are not entirely efficient is taken as fact by the great majority of the players in the trillion dollar industry that has grown around financial markets. This position might be biased, though, because if the financial industry were to accept that there is no way to "beat the market", i.e., to consistently obtain a return higher than that of a stock index fund with a simple buy-and-hold strategy, there would be no reason for hedge funds, brokers or even financial news networks to exist.

Given the numerous hypotheses stating that financial markets are completely random, it is possible that our attempt to develop a speculative, agent-based trading strategy might be considered a fool's errand by some. While it is undeniable that there is a lot of unpredictability and noise in financial price series, we do believe that the markets are not always efficient. As the stock market crash and the commodity bubble of 2008 have shown, financial markets can be far from rational at times, which negates the efficient market hypothesis (Fox, 2009). Many economists agree; Shiller (1992), for example, completely dismantled the efficient market hypothesis, and went as far as calling arguments in favour of this hypothesis "one of the most remarkable errors in the history of economic thought". From a trader's perspective, irrationality and inefficiencies in financial markets translate into profit opportunities. Our aim will be to research the development of intelligent agents able to exploit these opportunities.

1.2 Artificial Intelligence and Agency Terminology

Artificial intelligence is a very broad academic discipline, encompassing research topics like pattern recognition, search and optimization algorithms, planning, learning and reasoning techniques, among others. It is difficult to define the boundaries of this field, because it involves numerous unrelated methods and algorithms, such as:

- the A* search algorithm (Hart *et al.*, 1968), frequently used in pathfinding, i.e., determining the shortest route between two points;
- the expert system (Feigenbaum *et al.*, 1971), a system composed of a set of condition-action rules (the knowledge base) defined by domain experts, which can reason and answer questions by chaining those rules;
- the k-nearest neighbour classifier (Aha *et al.*, 1991), a lazy classification algorithm that will classify a test instance according to the classes of the training instances that are closest to it in the feature space;
- the AC-3 (Mackworth, 1977), an algorithm that solves constraint satisfaction problems, i.e., it finds solutions that satisfy a given set of restrictions;
- the hidden Markov model (Baum & Petrie, 1966), a temporal probabilistic model consisting of a finite set of states (each associated with a probability distribution) and a set of probabilities governing the transitions between these states; it has been extensively employed in speech, handwriting and face recognition;

- the minimax algorithm with alpha-beta pruning (Russel & Norvig, 2002), an algorithm that traverses search trees to find the next optimal move in a multiplayer game;
- the Graphplan (Blum & Furst, 1997), an algorithm for solving planning problems that outputs sequences of operations that will lead to the desired goal state;
- the genetic algorithm (Fraser & Burnell, 1970), an optimization algorithm that evolves candidate solutions (by mutating and combining the best) according to a fitness function.

These 8 examples are just a very small sample of the myriad of techniques that constitute the field of artificial intelligence. They showcase just how far-reaching the field has become, with researchers addressing many different types of problems, using completely different methods.

Defining what makes a machine "intelligent" is a controversial subject in artificial intelligence. Consider the case of chatterbots (Weizenbaum, 1966; Mauldin, 1994), i.e., bots that communicate with human users through text messages, used mostly on the Internet for advertising (and sometimes spamming) purposes. Contrary to what was expected several decades ago, their conversational skills are still very rudimentary; that is due to their algorithms being extremely naïve, something for which they have been harshly criticised in the academic world. When engaging a human in a conversation, chatterbots mostly resort to tricks – text pattern matching, intentional typos, rude language – to attempt to mimic human behaviour, and deceive the human user into thinking they can understand the conversation. If they succeed, should we consider them intelligent? Searle (1980) says we should not, and describes a scenario – the Chinese room experiment – that exposes the difference between that "fake intelligence" and "real intelligence". He pictures being in a closed room, and receiving Chinese messages from people on the outside; by using a basic set of tricks and rules (which equate to a computer program), he should be able to fabricate sensible replies to some of those messages, leading the people he is communicating with to believe that he speaks Chinese, when in fact he does not. Clearly, there is a difference between understanding Chinese and simulating the ability to understand Chinese. Searle concluded the latter is not true intelligent behaviour, because there is no thought process behind it. This reasoning is the main argument against the Turing test (Turing, 1950), and contributed to its decline as a possible metric for machine intelligence – according to the proponents of this test, a machine could prove it was intelligent by chatting with human users, and fooling them into believing they were having a conversation with another human. The best chatterbots are still far from passing the Turing test; because of their naïve strategies, the majority of the AI community sees them as gimmicky and detrimental to the field. Nevertheless, we appreciate Turing's position on the matter: if we focus solely on the way the machines act, the distinction between real and simulated intelligence becomes irrelevant; people want to see machines that act intelligently, regardless of how they do it.

By Searle's definition, to be truly intelligent, a chatterbot would need to be capable of processing and understanding natural language. This, however, is an AI-complete problem. Solving it would imply creating machine intelligence as it is portrayed in science fiction, i.e., artificial intelligence that matches or exceeds general human intelligence (strong AI). Even though some tiny advances have been described in a few on-going experiments (Markram, 2006), researchers are still far from reaching that objective. For all we know, it might even turn out to be unreachable – there is no way to tell if the biological process behind human intelligence can be replicated with a digital machine (Dreyfus, 1979; Searle, 2004). But there are optimists in the field. Kurzweil (2006) is one of several futurists who believe that machines showcasing strong AI are just a few decades away. His enthusiasm reminds us of the unbridled optimism of the 1960's, when many predicted machines would soon be able to do anything a human being could do (Simon, 1965; Minsky, 1967). Kurzweil's forecast seems destined to fail the same way, considering the current state of the art, and also the numerous geopolitical and economic challenges that humanity might face in the coming future (a factor that is often disregarded in these far-reaching predictions). If we look at the current state of the field, we can verify that the most advanced artificial intelligence techniques do not even come close to addressing the complexity that characterises general intelligence, something that biologists themselves have yet to grasp. Nevertheless, even if these techniques are not yet powerful enough to be utilized in the implementation of artificial general intelligence, they do allow for the creation of intelligent machines that outperform human experts at specific tasks. IBM's chess-playing computer Deep Blue[1] proved this point, when it defeated world champion Garry Kasparov in 1997. Nowadays, the application of artificial intelligence to perform concrete tasks is so pervasive that it often goes unnoticed. This is known as the "AI effect": as soon as a machine is able to do something that was previously thought to require some sort of intelligence, that ability starts being taken for granted by its users, which will no longer consider it true intelligent behaviour. Voice recognition in cell phones, face detection in digital cameras, email spam filtering, query matching in web search engines, medical diagnostic systems, fraud detection systems, cruise control in vehicles, these are just a few of the countless ways in which AI has become an important part of our everyday lives.

One field where the use of artificial intelligence has been gaining momentum is financial engineering. S&P's Neural Fair Value 25 portfolio[2], for example, is a well-known practical application of AI; it lists the 25 stocks with the biggest price appreciation potential, picked weekly from a universe of over 3,000 stocks using artificial neural networks. Also, in the algorithms' "arms race", it has been reported that many hedge funds are now utilizing data mining and other AI methods to perform quantitative analysis (Davidson, 1997; Duhigg, 2006; Patterson, 2010; Yamazaki & Ozasa, 2011), although little is known about their proprietary setups. We should point out that, just because a hedge fund says its strategies are AI-based, that does not necessarily mean that it can select portfolios better than "a blindfolded monkey throwing darts at a newspaper's

[1] Information on IBM's Deep Blue is available at http://www.research.ibm.com/deepblue/.

[2] The Neural Fair Value 25 portfolio is published at http://outlook.standardandpoors.com.

financial pages" (Malkiel, 1985). Despite all the hype and fascination that surrounds artificial intelligence in traditional media, it obviously does not confer machines any supernatural powers, so it is important to remain sceptical when it comes to the outrageous claims of some financial services providers. Our research will attempt to shed some light on this matter – we want to determine exactly what may be expected from intelligent machines when inserted in financial markets. In addition to machine intelligence, we will also be focusing on machine autonomy. That is to say, the focus of our research will be the deployment of intelligent agents in the financial industry. Even though the field of agency is relatively new, it has already become an important branch of computer science; it has connections to several areas of research, among which economics, game theory, distributed systems, and even psychology. It is also intrinsically connected to the field of artificial intelligence, so much so that intelligent agents are mentioned in some tentative definitions of AI. Russel and Norvig (2002), for example, define artificial intelligence as:

"… the study of agents that receive percepts from the environment and perform actions." (p. vii)

Defining intelligent agent is a bit harder, because there is no consensus regarding the characteristics that these agents should exhibit. We believe Wooldridge's (2002) definition is as good as any; it puts the emphasis on what we consider to be the two most distinguishing traits that intelligent agents should possess:

"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." (p. 15)

In other words, an agent is an entity that acts autonomously and that exhibits goal-oriented behaviour. Notice this definition also fits entities that would not normally be associated with intelligence. For instance, web crawlers are able to index entire websites automatically, but that does not make them intelligent; likewise, a thermostat is capable of controlling the temperature of a system, but the systematic strategy it employs to achieve that goal cannot be considered intelligent behaviour. The way we see it, the distinction between these and truly intelligent agents should be made based on the complexity of the task being performed. In order to make this distinction less subjective, other qualities have often been attributed to intelligent agents, including the ability to learn in real-time (Franklin & Graesser, 1996), and being proactive and capable of social interaction (Wooldridge & Jennings, 1995), i.e., capable of communicating with other agents and entities.

Regarding the implementation of intelligent agents, several agent architectures have been suggested throughout the years. According to Maes (1991), an agent architecture is:

"A particular methodology for building agents. It specifies how … the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions … and future internal state of the agent." (p. 115)

Classic agent architectures are based on symbolic AI, meaning the agents' knowledge about the world is represented explicitly with facts and rules. Using this knowledge, the agents decide when and how to act through logical reasoning, i.e., via theorem proving. Several classic architectures follow the BDI software model of agency; in this model, the agents' behaviour is regulated by explicit symbolic representations of their "mental attitudes": beliefs (what they think they know about the world), desires (their objectives) and intentions (the decisions they are committed to take). PRS (Georgeff & Lansky, 1987) and IRMA (Bratman *et al.*, 1988) are examples of this type of architecture. IRMA agents, like many other symbolic AI-based classic agents, are planners: knowing the potential effect of each of their actions to the environment, they select the course of action (i.e., the plan) that is expected to take them closer to the objective. SOAR (Laird *et al.*, 1987) and ACT-R (Anderson, 1996) are other examples of symbolic agent architectures in which the agents' behaviour is governed by explicit production rules. Several agent programming languages have been proposed for developing this type of agent, whose reasoning relies on pattern matching and symbolic processing. These include, among others, the concurrent MetateM language (Fisher, 1994), with which the agents are programmed using temporal logic, the Golog (Levesque *et al.*, 1997) and GOAL (Hindriks, 2001) languages, which are based on Prolog, and the 3APL language (Hindriks *et al.*, 1999), used for implementing agents with beliefs, desires (declarative goals) and intentions (procedural plans).

The classic approach of looking at intelligent agents as simple theorem provers or expert systems presents a few serious limitations, among which the difficulty in translating the real world into a symbolic representation that is comprehensive enough, and having the agents process that representation and reason in time for their decisions to be useful (Wooldridge & Jennings, 1995). The idea that intelligence can be deconstructed into explicit representations of knowledge is also arguable. Brooks (1990) postulated a different type of intelligent behaviour, based on reactions rather than logical reasoning. He argued that a system can act intelligently without a symbolic representation of knowledge, and postulated that systems can only demonstrate intelligence when grounded in the physical world. He proposed a method for building intelligent agents named subsumption architecture; unlike the classic approach, where the combination of interconnected reasoning modules commands the agent's conduct, the subsumption architecture consists of a layered hierarchy of simple independent behaviour-generating modules which compete to dictate the agent's actions. Once the agent is placed in the physical world, its intelligent behaviour emerges from the competition between these modules. Brooks co-founded the iRobot company, maker of intelligent robots like the famous Roomba vacuum cleaner[3]; this cleaning robot does not keep any information about the room it is vacuuming; instead, it reacts to the environment as it moves, changing its behaviour whenever an obstacle is hit. From a practical stance, this may be considered intelligent conduct, which validates Brook's belief that the intelligence of an agent should be judged according to its actions, regardless of the reasoning underlying those actions. When

[3] Info on iRobot Corporation's Roomba is available at http://www.irobot.com.

comparing the symbolic approach, more rooted in theory, with his bottom-up agent building strategy, Brooks maintained that the practical usefulness of the agents is what ultimately mattered:

"A further part of our strategy then, is to build systems that can be deployed in the real world. At least if our strategy does not convince the arm chair philosophers, our engineering approach will have radically changed the world we live in." (p. 13)

Obviously, Brooks' reactive approach and the classic deliberative approach are not mutually exclusive. In fact, they seem to complement each other nicely. Ferguson's TouringMachines (1992) is an example of a hybrid agent architecture that attempts to combine the two philosophies.

Brooks' contribution to the field of agency deserves special credit, because he was able to come up with a practical application for his research. In the middle of the 1990s, many researchers thought intelligent agents represented a paradigm change, and believed their utilization in commercial and industrial settings would soon become widespread. Personal digital assistants (Maes, 1994), for example, were expected to take over the Internet. However, more than a decade later, this revolution has yet to occur, mainly because too much time was spent coming up with theoretical solutions for abstract problems that bear little resemblance to real world problems, and little was spent addressing the practical issues surrounding the deployment of useful agent-based production systems. Hendler (2007) describes the current state of things very succinctly:

"While there's clearly still an active research community in agents, I see no evidence for the imminent widespread use of this technology such as we were promising a decade ago … The bulk of the papers I can find published since then are filled with all kinds of wonderful theory but not much on deployed applications … I ask again: Where are all the agents?" (p. 3)

Still, there are a few real life applications of agent technology worth mentioning. For instance, the sales numbers of the aforementioned Roomba vacuum cleaner are in the millions. The technology's enormous potential is also showcased in the newest generations of humanoid robots; Honda's

ASIMO [4] can act autonomously with concrete goals, applying several artificial intelligence techniques to, among other things, recognize moving objects and faces, distinguish sounds, move in circular patterns, go up and down stairs, and run at up to 6 km/h. In regard to software intelligent agents, there are also some very impressive applications. IBM's Watson[5], for example, has been creating a lot of stir lately; with its enormous knowledge base, and its ability to "understand" natural language, this agent was able to beat the best human participants in the television quiz show Jeopardy! (Baker, 2011). Another amazing commercial application of agent technology is the MASSIVE 3D animation software[6], originally developed by Stephen Regelous. MASSIVE, short for Multiple Agent Simulation System In Virtual Environment, is a software package that simulates crowd-related visual effects. It can create millions of individual agents, each with the ability to act autonomously, according to a loosely defined set of parameters. MASSIVE has been used to produce special effects for several TV ads and shows, as well as blockbuster movies like "The Lord of the Rings" and "300". Each agent created by MASSIVE is, from a practical point of view, another actor at the orders of the movie director.

The MASSIVE software showcases the potential of multi-agent systems, i.e., distributed systems in which multiple autonomous intelligent agents interact. Peer-to-peer communication in a multi-agent system is accomplished with an agent communication language, which specifies the syntax of the messages that may be exchanged. The first such language to gain relevance was the KQML (Finin *et al.*, 1994), now made obsolete by the FIPA-ACL (FIPA, 2002), the communication language born out of the efforts of the Foundation for Intelligent Physical Agents. As for the implementation of the multi-agent systems, there are currently numerous commercial and open source software packages available. Some of these packages will not only take care of the

[4] Honda's ASIMO humanoid robot is shown at http://world.honda.com/ASIMO.

[5] IBM's Watson program is described at http://www-03.ibm.com/innovation/us/watson/.

[6] The MASSIVE software by Massive Software is available at http://www.massivesoftware.com.

agents' communication and interactions, but also facilitate the development of the intelligent agents. IBM's ABLE (Bigus *et al.*, 2002), for instance, is a Java open source modelling toolkit that expedites the implementation of multi-agent systems composed of hybrid intelligent agents, whose behaviour is dictated by rule-based reasoning and machine learning. Other Java packages include the JADE (Bellifemine *et al.*, 1999), a FIPA-compliant software framework for multi-agent systems, and Cougaar (Helsinger *et al.*, 2004), a research project of the U.S. Department of Defense.

The way agents interact in a multi-agent system will vary according to their objectives. Agents might need to compete with each other to pursue their individual goals, or they might need to cooperate with one another, to optimize the performance of the system as a whole. For each scenario, there needs to be a specific protocol regulating the interactions between them – this is known as the negotiation protocol, and defines the "rules of encounter" between agents (Rosenschein & Zlotkin, 1994). For each negotiation protocol, there is usually an optimum negotiation strategy that the agents should use. Take the English auction of a good or service, which is an example of a competitive scenario. The negotiation protocol in this system is as follows: each agent can bid more than once; when bidding, an agent must offer more than the current highest bid; if no agent is willing to raise the bid, the good is allocated to the agent that made the last bid, and the auction ends. The best negotiation strategy that an agent could follow in this scenario is to continuously increase the bidding price using small increments, until the bid reaches the price that it believes the good is worth, at which point it should stop bidding (Wooldridge, 2002). For an example of a negotiation protocol in a cooperative scenario, we can look at the monotonic concession protocol. It is meant for the negotiation between two agents, and defines the following rules of interaction: the negotiation is done in rounds; in each round, both agents propose a deal; agreement is reached when one of the agents determines that the deal it was offered is at least as good as its own proposal, according to its utility metric; when this happens, the deal is accepted and the negotiation ends. The best course of action for agents in this scenario is to initially propose their most preferred deal, and then proceed to make concessions based on how much they are willing to risk conflict (Rosenschein & Zlotkin, 1994). This strategy will ensure that they achieve Nash equilibrium (Nash, 1950). Nash equilibrium is a solution concept from game theory that defines a situation in which none of the players has anything to gain by changing its strategy unilaterally. It is common to find game theory concepts in multi-agent systems' research, because these systems describe the classic case study for game theorists: a social setting where the success of each entity/agent is affected by the choices of others.

We have now presented the most important concepts in the field of agency. With a little imagination, we can easily draw a parallel between intelligent agents, and some of the entities that populate the world of finance. Specifically, we can imagine the agents playing the part of financial traders, grouped together in multi-agent systems that act as autonomous hedge funds. These agents would interact (with a specific agent communication language) and attempt to agree on the trading decisions (with a negotiation protocol), so as to work towards the greater good of the multi-agent system. This idea will be the theme of our research.

1.3 Objectives of the Research

It is our belief that multi-agent systems are well suited for financial trading. Assuming the ability to trade profitably is a real skill, and not just the result of luck, there are several characteristics that could give software intelligent agents an advantage over their human counterparts. For example, unlike human traders, the agents can trade 24 hours a day; this feature is particularly important if the target market is the foreign exchange, because this market is continuously open 5 days a week. Also, the agents should be able to make trading decisions much faster than humans, and will not be influenced by fear or greed (unless these emotions somehow emerge from their implementation); this should help them outperform human traders whenever the market enters volatile and stressful periods, because their judgement will not be clouded by those emotions. Finally, software agents should be much easier to manage and control, because their loyalty, honesty and obedience are never an issue. This is an important advantage. Many financial institutions have experienced massive losses due to the destructive actions of a single rogue trader. The most famous example is probably the bankruptcy of Barings Bank, which resulted from unauthorized trades by one of its traders, Nick Leeson (Leeson & Whitley, 1996). The fact that intelligent agents are always "honest", work faster, and do not require compensation or vacation time, suggests that they could become an important part of the financial industry.

There is no clear-cut way to create these trading agents. Every solution will require a bit of guesswork, and will reflect the researchers' own views on what successful trading is, and what it entails. Our method will be based on a mixture of artificial intelligence technics, with which we will design a custom-made architecture meant for the development of intelligent agents capable of negotiating any type of financial instrument; we will also propose negotiation protocols for regulating their interactions in multi-agent systems. To ensure that our solution is practical, and may be employed in real life, we will use it to develop trading agents that will be tested with real life data, in lifelike conditions. When analysing their suitability for the task at hand, we will consider not only their trading results, but also their ability to exhibit intelligent behaviour, i.e., their ability to adapt to market changes, to stop trading under adverse conditions, and to limit the risk as much as possible.

The main objective of our research has now been defined: we will try to demonstrate that intelligent agents are a good substitute for human traders. But what exactly is the point of this type of study? Ultimately, we are talking about financial speculation, an activity that many would frown upon. Speculators are often depicted as gamblers and leeches on society (Angel & McCabe, 2009), so speculation is probably not the most righteous topic for scientific research. Granted, this is not the noblest of activities, but one may argue that speculators do play an important role in financial markets by providing liquidity (Volpe & Dickson, 2004), without which the markets would collapse or grind to a halt (Brunnermeier, 2009). Whether we like it or not, financial speculation is at the core of a very important services industry, one that will keep affecting us all in the foreseeable future. Putting these ethical concerns aside, we believe that this industry should have a significant interest in intelligent agents that can trade successfully. Notice this concept of "successful trading" is subjective – it depends on how much risk one is willing to accept for a given expected return. We will be proposing our own definition of what makes a trader successful; if our intelligent agents are able to satisfy the requirements of that definition, they should be capable of trading safely and profitably in the long run; by doing so, they will empirically prove that agent technology can be usefully deployed in financial settings – this would be an important contribution to a field that, with regard to practical applications, has often been criticized for over-promising and under-delivering.

Before proceeding with our research, it is important to stress that the feat we are trying to achieve is far from trivial. Financial markets are usually associated with "easy money" by the general public, mostly because of the way the media reports on these markets – they have this casino mentality of praising the (few) winners and ignoring the many losers. Yet, the idea that profitable financial trading is easy could not be further from the truth. Take the Barclay Hedge Fund Index[7], which tracks the average net return of several thousand hedge funds. These entities are supposedly the most sophisticated participants in financial markets: they are mostly unregulated, and have broad flexibility in the type of positions they can hold, and the type of financial instruments they may invest in. They trade other people's money, with the fund manager usually collecting an annual management fee and a performance fee; a common fee schedule is the "2 and 20", meaning an annual management fee of 2% of the fund's net asset value, and a performance fee of 20% of the profits. Being this expensive, and given all the flexibility they have, one would expect these financial

[7] The Index is available at http://www.barclayhedge.com/research/indices/ghs/Hedge_Fund_Index.html.

institutions to be able to achieve above average returns with relatively low risk. The average annual return for the several thousand hedge funds tracked by the Barclay Index is shown in Figure 1; also plotted in this chart is the average annual yield of the 10-Year U.S. Treasuries (government debt of the United States of America), considered by many to be a riskless investment, and the annual return of the S&P 500 Index (including dividends), which is a capitalization-weighted index of the prices of 500 large-cap stocks traded in the NYSE and the NASDAQ stock markets. We can see in this figure that the hedge funds had an average net return of -21.6% in 2008; this statistic alone should scare the most risk-averse investors from putting their money in these institutions. But let us disregarded the risk, and concentrate solely on the profit. Maybe the hedge funds' return justifies all the risk they incur. According to the Hedge Fund Index, the average annual return of the hedge funds in the last decade (from 2000 till 2009) was 8.3%. This is an acceptable profit, considering the average annual yield of the "risk-free" investment (the 10-Year U.S. Treasuries) throughout the same period was 4.5%, and the average annual return of the S&P 500 Index was just 1.2%. Similar comparisons are often publicized in financial media, leading viewers to believe that hedge funds are mostly profitable, and will easily achieve above average returns. The problem with this reasoning is

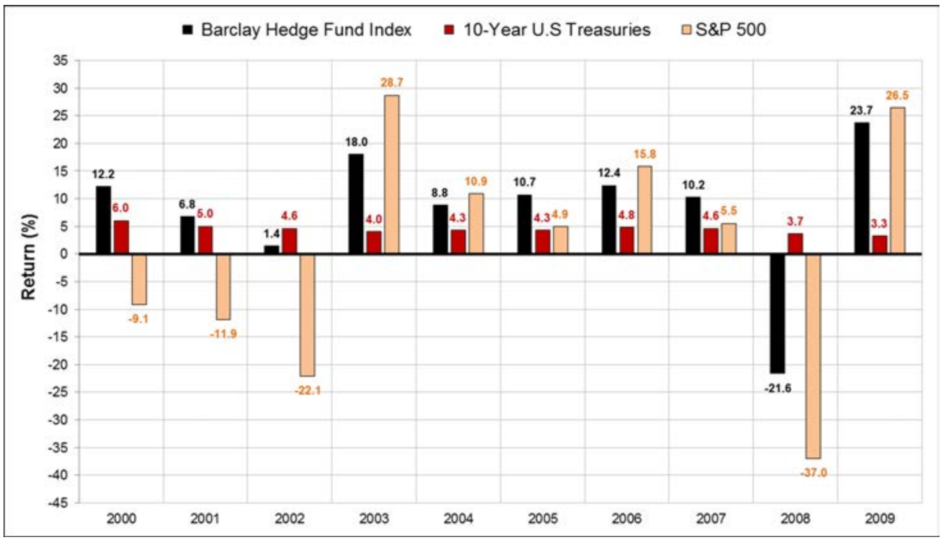


Figure 1. Net annual return of the Barclay Hedge Fund Index, the 10-Year U.S. Treasuries and the S&P 500 Index.

that the values in the Barclay Index are artificially inflated. Like all other hedge fund tracking indices, the Barclay Index suffers from severe survivorship bias, because hedge funds that drop out of the index usually do not report their final losses. Since 2005, more than 20% of the hedge funds (by assets) stopped reporting to Barclay Hedge, either because they were liquidated, or because their managers decided to stop disclosing the returns. That percentage spiked to 38% in 2008, a year that was clearly bad for most hedge funds, due to the increased volatility in financial markets caused by the subprime mortgage crisis. The losses that led these hedge funds to stop reporting their returns were rarely disclosed, meaning they were never reflected on the value of the hedge fund tracking indices. TrimTabs, an independent research firm, published a study in 2009 showing the real impact of this type of bias in the Barclay Index. As is, the index indicates that the hedge funds' average return from January 2005 to June 2009 was 25.4%. According to the TrimTabs research, if we assume a reasonable 30% loss rate for the funds that stopped reporting during that period, the average return drops to -3.4%. Another study by Horst and Verbeek (2007) suggests that different types of bias could be inflating the average returns in hedge fund databases by as much as 8% per year. In addition to this, we should point out that the hedge fund industry is not exactly known for its honesty. Some funds are outright Ponzi schemes (Officer, 2009), or engage in fraudulent activities like front running or insider trading (McCool, 2010). Others overstate their returns (Bollen & Pool, 2009). Finally, there are those that, either purposely or naïvely, pursue trading strategies with Taleb distributions (Kay, 2008; Wolf, 2008). These strategies are best described with the analogy of "picking up pennies in front of a steamroller": basically, the fund managers are making trades that have a high probability of producing small gains, and a low probability of producing very large losses; until an outlier event occurs, the managers obtain steady returns with the appearance of very low risk, which helps them raise more investment capital from outside sources, which in turn translates into bigger commissions and fees; however, when the tail event inevitably happens, their strategies are "steamrolled", and the funds experience massive losses.

More often than not, these hedge funds will vanish into thin air, hiding the losses from all but the fund investors, who end up losing most of their money, while the managers get to keep the fees collected up to that point. This all goes to show that, despite what one might think when looking at hedge fund tracking indices and listening to mainstream financial media, it is doubtful that the average hedge fund can offer above average returns in the long run (Dichev & Yu, 2010). As we see it, the fact that the (allegedly) most professional financial players have trouble returning a decent profit in a consistent manner proves that successful trading is an extremely difficult endeavour. Hence, our intent to develop a method for implementing intelligent agents that can accomplish this feat is, undoubtedly, a complicated proposition. On the plus side, since this is such a difficult task, there should be plenty of interest in one such method, if the agents are able to achieve acceptable results consistently. This interest is just one of the reasons why we believe our research will be valuable; we also expect to contribute to the advancement of artificial intelligence and agent technology, by designing a new framework that is completely based on the extensive body of works available in both fields, and then applying it in one of the most competitive industries in the world.

1.4 Overview of the Thesis

The research reported in this thesis is interdisciplinary, and revolves around two main topics: artificial intelligence (particularly data mining and agent technology) and financial trading. In Chapter 2, we will be looking at the current state of the art regarding the application of artificial intelligence techniques in the trading of financial instruments. We will show that, while there are many studies describing the use of data mining models to perform financial time series prediction, there is not much literature available on using intelligent agents as autonomous traders. That is the void that our research intends to fill.

In Chapter 3, we will describe a novel hybrid cognitive architecture for implementing intelligent agents with the ability to trade different types of financial instrument. Trading agents based on this architecture are expected to be capable of maximizing the profit, while simultaneously attempting to minimize the risk. The construction of the architecture will be explained step-by-step, with all design decisions being subject to critical analysis.

Chapter 4 will start with a brief description of all the data mining models and attributes that we intend to use in the implementation of the trading agents. This will be followed by the description of 10 currency trading agents, developed according to the proposed architecture. These agents will be tested with out-of-sample data in lifelike conditions, and will later be integrated in a multi-agent system, for which we will create an agent communication language and a negotiation protocol. This system will be tested with the same out-of-sample data, simulating trades for a period of around 2.3 years.

In Chapter 5, we will analyse the architecture's suitability for trading another type of financial instrument: we will utilize it to implement 25 stock trading agents. These agents will be put to the test, individually and as part of a diversified investment strategy, by simulating trades with 3.3 years' worth of test data. Additionally, we will describe a system that allows the forward-testing of the agents: it publishes their trading decisions every day in a public website, making it possible to study their behaviour as time goes by; this means that the analysis of the agents' performances is not limited to the results presented in this thesis – their current trading activity is available online, and can be followed in real-time.

Chapter 6 shows the results of some experiments we did using index data. We implemented several new agents, and had them trade for extended periods of time and with bigger time frames, in order to examine their behaviour over the long run. In the last part of this chapter, we describe the use of two different resource allocation strategies (leveraging and compounding) to improve the performances of these agents.

In Chapter 7, we present the culmination of our work: the implementation of a sizable multi-agent trading system that integrates all the currency and stock trading agents that were developed.

With this system, we intend to demonstrate a specific potential application for agent technology in the investment industry: the creation of AI-based hedge funds where human intervention is kept to a minimum. We will list the reasons why these systems could become an important part of the industry, and will try to obtain some meaningful simulated trading results, to get an idea of what sort of performance may be expected from them.

Finally, in Chapter 8, we will do one last analysis of the agents' results, and draw our conclusions regarding their usefulness. We will also suggest further improvements that could be made to their implementation, and propose future research to be done on this topic.

Chapter 2

Artificial Intelligence in Financial Trading

Quantitative analysts, or quants, are responsible for coming up with complex mathematical and statistical models, and using them for several investment-related tasks, among which risk management, derivatives pricing and financial trading. The technological boom of the last decades has made quants' job much easier – it is now simpler than ever to process and model large amounts of financial data with powerful computers. A study by the Aite Group (2008) estimates that as much as 12% of all assets under management in the world were driven by quantitative analysis in 2007, a figure that was expected to increase to 14% in 2010. In addition to being applied in the development of quantitative investment models, computers are increasingly being put in charge of making the trading decisions themselves. That is to say, they decide all the details of the trades (timing, quantity, entry and exit prices, etc.), and automatically send the orders to the markets, thus completely doing away with human traders. This strategy is known as algorithmic trading. In the past few years, it has completely revolutionized the investment landscape, to the point where computers are now the most influential players in some markets. For example, the Tabb Group (2009) estimates that high frequency trading is now accounting for more than 60% of all equity share volume in U.S. stock markets; high frequency trading is a special type of algorithmic trading, characterised by the buying and short selling of huge amounts of shares, with a very short time frame (milliseconds), and often associated with questionable strategies (Arnuk & Saluzzi, 2009). It is fair to say that, as time goes by, quantitative and algorithmic methods will become even more dominant in financial markets. Artificial intelligence could turn out to be an important catalyst in this process, given that some AI techniques are perfectly suited for performing quantitative analysis. Data mining models, for instance, can easily find hidden patterns in large amounts of financial data, and these patterns can in turn be utilized to create algorithmic trading strategies. Several quants and AI researchers have already begun investigating this subject. However, artificial intelligence has also been receiving a lot of attention from dubious sources: many so called AI-based trading bots and strategies are being sold online with the promise of unrealistic returns, as the sellers try to capitalize on the public's misconceptions about artificial intelligence. While there is no question that data mining models can find patterns in financial data much faster (and better) than human traders ever could, that does not mean that creating profitable trading strategies with these models will be easy, if at all possible. Data mining will only be useful if there are any predictive patterns in the financial data to begin with, something that the efficient market hypothesis completely rejects. But even if the models find relevant patterns, that still does not mean that we will be able to use them to trade profitably, because financial trading entails considerable costs. Thus, it is important to separate the myth from reality: artificial intelligence is not the be all and end all of financial trading. Nonetheless, it does serve its purpose, as many studies have already proven. Looking at the literature that is currently available, we divide these studies in three categories, based on the complexity of the task being researched. The simplest studies are those that describe the use of a single data mining model to make financial predictions; from a practical point of view, these are simple tools meant to aid human traders in their decision process. The second type is slightly more complex; instead of a single model, an ensemble of data mining models or a hybrid system are tasked with doing the forecasts. Finally, there are a few studies that describe

AI-based trading systems intended to replace human traders altogether; these are the most relevant to our work. We will examine several of these studies in the sections that follow.

2.1 Financial Prediction with Standalone Data Mining Models

One of the simplest ways to develop a trading strategy for a particular stock is to model its price time series. If the model captures the essence of the underlying data generating process, it will be able to output accurate predictions regarding the stock's future price, and we can utilize these predictions to open trades. The autoregressive moving average (ARMA) model, developed by Box and Jenkins (1976), is a classical statistical tool that has been employed for many decades in the modelling of time series; it is called autoregressive because it uses past values of the series, also known as lagged values, to predict future values. Another time-tested statistical model is the autoregressive integrated moving average (ARIMA), an adaptation of the ARMA model for non-stationary time series (i.e., series whose statistical properties vary with time); this model includes an initial step that differences the data, to make it stationary. Given the widespread use of these two models, they are frequently utilized as a benchmark to evaluate the performance of more complex nonlinear data mining models. Wu and Lu (1993), for example, implemented a stock market forecasting system using artificial neural networks, and compared its performance with that of an ARIMA model. The objective of their system was to predict, on a daily basis, if the value of the S&P 500 Index was going to increase, decrease or remain unchanged. They tested the system, and concluded it performed better than the ARIMA model, when the market was stable. However, their results do not support the claim that artificial neural networks can outperform ARIMA models: the system's accuracy was 23%, while the ARIMA's was 42%. This is a rather ancient article, so it is possible that the lack of computational power at the time might have hindered the researchers' objective (the training of artificial neural networks is quite demanding). A more recent study by Kamruzzaman and Sarker (2003) shows more encouraging results. They compared the performance of an ARIMA model with that of several artificial neural networks, trained with different algorithms: backpropagation, scaled conjugate gradient and backpropagation with Bayesian regularization. Their goal was to predict the weekly exchange rate of several currency pairs; the inputs to the neural networks were the exchange rate in the previous week, and moving averages of prior weekly exchange rates; the performance was measured using the normalized mean square error (NMSE), the mean absolute error (MAE) and the accuracy predicting the direction of the price (DS). Their experiments showed that all the neural networks could outperform the ARIMA model, regardless of the metric. However, the results they got seem too good to be true: the neural networks predicted the direction of the price with close to 80% accuracy in their tests. Exchange rates cannot be that predictable, as that would imply that the Forex market is extremely inefficient – and we are certain it is not. A closer look at the researchers' testing method reveals what made that unbelievable accuracy possible: several neural networks of each type were trained, but only the best ones were discussed in the study. Since the models were chosen based on how well they performed with the test data, we can conclude that the results are biased – more than likely, the artificial neural networks would not be able to exhibit the same level of accuracy with new unseen data. Avoiding this type of bias is extremely important when creating a trading strategy with a data mining model, because it will cause disastrous losses in real life.

There are many other articles on the subject of using artificial neural networks to model financial data. Saad *et al.* (1998) used time delay, recurrent and probabilistic neural networks to predict if the price of 10 stocks was going to increase at least 2% in the subsequent 22 work days; inputs to the neural networks were all based on the stocks' daily closing prices. Their results showed that the three types of neural networks outperformed a Fisher linear classifier, by finding more profit opportunities and avoiding more false positives. Accuracy-wise, the results were quite impressive; for Apple's stock, for example, the accuracy of the three neural networks was greater than 90%. But once again, we find some flaws in the experiments that originated these results. First of all, the datasets used for testing were relatively small, varying from 100 to 200 instances, and the total number of profit opportunities predicted by the neural networks was also too small; because of this, it is impossible to extrapolate how well these models would behave if we employed them to trade over a long period of time (especially in periods of high volatility). Secondly, they tested several neural networks with different parameters, but only reported the results of the best ones; as previously mentioned, this procedure taints the results – the extraordinary accuracy they got is probably bounded to the specific set of test instances with which the models were evaluated, and for which their settings were optimized. Also, in regard to the usefulness of these models in practice, the study's results are inconclusive: they provided the accuracy, but there is no indication of how much profit one would get with the accurate predictions, compared to the losses suffered with the bad forecasts. In real life trading, high accuracy does not necessarily translate into big profits, because a single inaccurate prediction can yield a large loss that wipes out the profit of several successful trades. Nevertheless, the researchers concluded that it is possible to predict short term trends using artificial neural networks trained with historical closing prices. Most applied AI studies on this subject show similar empirical evidence against the efficient market hypothesis. However, it is important to ensure that the evidence does not arise from biased results, or from their incorrect interpretation; this is tricky at times, because performance analysis is not always straightforward. We encountered this issue in a study by Zhang *et al.* (2002). They used granular and backpropagation neural networks to predict the prices of six different stocks; the inputs to each network were the open, high and low prices for each stock on a given day, and the output (the prediction) was the closing price the following day. After testing, they concluded that the granular neural networks performed better and faster than the backpropagation neural networks. However, because they utilized the average error to measure the performance of the models, not much can be said about their utility for predicting future prices. For example, the average error predicting the price of the Dow Chemical stock was $1.39 with the granular neural network, and $3.38 with the backpropagation network. Since they did not provide the variance of the price, we cannot tell if this is a good or a bad performance – if the average change in the test period was $10, we could say that the models worked well; but if was just $1, their predictions would be worthless. As Swingler (1996) points out in an article that lists common pitfalls in financial prediction, the mean squared error (MSE) and the average error metrics are only helpful if accompanied with volatility information.

Tenti (1996) compared data mining models based on the returns that could be obtained with their predictions, which makes the results much easier to interpret. As we see it, the return is one of the best metrics for this type of research. He used three recurrent neural networks to predict price changes of currency futures, and implemented two trading strategies with them; the inputs to the networks included lagged returns and technical indicators such as the average directional movement index (ADX), the trend movement index (TMI) and the rate of change (ROC). After taking into account the trading costs, all the recurrent neural networks achieved positive returns in the trading simulation. With one of the strategies, the networks' yearly unleveraged returns were 3.9%, 8.6% and 27.7%. This supports our a priori expectation that the financial markets are not completely efficient, and that some small inefficiencies may be exploited for a profit. While not very impressive, Tenti's unleveraged returns are in line with what we believe to be feasible with an AI-based trading strategy, considering the difficulties associated with successful financial trading.

So far, we have only looked at articles describing the use of artificial neural networks in financial forecasting; going by the volume of publications, these nonlinear models seem to be the most preferred tool for mining financial data. Nevertheless, there are a few studies reporting results obtained with other types of models. Kim (2003) used support vector machines to forecast the direction of the daily price of the KOSPI stock index; the inputs to the models consisted of 12 technical analysis indicators, among which the momentum, the Williams %R and the commodity channel index (CCI). Several support vector machines were trained, using different parameters; the best model achieved an accuracy of 57.83% forecasting the test data. For comparison purposes, several backpropagation neural networks and nearest-neighbour models were also trained. The best neural network had an accuracy of 54.73%, while the best nearest-neighbour model had an accuracy of 51.98%. This precision is not very good, especially taking into account that it refers to the best models that were trained. But realistically speaking, this is exactly what we should expect, because the noise and the randomness in financial data make it extremely difficult to predict. A study by Tay and Cao (2001) also showed that support vector machines can, at times, perform better than backpropagation neural networks; while forecasting the value of five different futures contracts, the support vector machines got, on average, better results than the neural networks, according to several metrics; for instance, predicting the direction of the price, the support vector machines achieved a mean accuracy of 47.7%, versus 45.0% for the neural networks. Chen *et al.* (2006) reached a similar conclusion: based on five different metrics, their research revealed that support vector machines could outperform backpropagation neural networks in the prediction of the value of six Asian indices; the average accuracy forecasting the indices' direction was 57.2% with the support vector machines, and 56.7% with the neural networks.

Andriyashin *et al.* (2008) tested decision trees to select stock portfolios; they used classification and regression tree models (CART), trained with attributes from fundamental analysis (earnings per share, sales, dividends, etc.) and from technical analysis (moving averages, momentum, rate of change, etc.), to classify German stocks. The classification reflected the weekly position to take: long, short or neutral. Using the suggestions from the trees, they were able to get a promising annualized return of 19.99% after expenses in the out-of-sample period. However, each tree classified less than 100 instances, and the overall return curve seemed somewhat correlated with the direction of the market, so it is hard to tell how these classifiers would fare going forward.

In a comparison study featuring models of different types, Zemke's (1999) demonstrated that a nearest-neighbour model could outperform a backpropagation neural network and a naïve Bayes classifier in the weekly prediction of the direction of the WIG stock index. After tuning the models' parameters (which might have tainted the results) the best accuracy achieved with the nearest-neighbour model was 64%, while the best accuracy with the neural network was 63%, and 54% with the naïve Bayes classifier. Rodríguez *et al.* (1999) confirmed the usefulness of nearest-neighbour models in financial forecasting: predicting one-day-ahead exchange rates for several currencies, these models achieved an average accuracy of 59.6% forecasting the direction of the rate, versus 56.7% with an ARIMA model. We should note that, while these articles might provide some insight regarding which models are better suited for financial forecasting, all comparisons must be taken with a grain of salt – the results will always depend on how much time was spent fine-tuning the parameters of each model, and selecting its training attributes.

Genetic algorithms, traditionally applied in optimization problems, have also been utilized in financial forecasting. The most straightforward way to do this is to make the algorithm optimize a rule-based trading strategy, as described by Mahfound and Mani (1996). Starting with a random population of simple trading rules, these researchers used the genetic algorithm to evolve the population, until they got an optimized set of rules. These rules indicated if a stock should be bought, sold or if no action should be taken, with a time frame of 12 weeks. In cross-validation, they got an accuracy of 87.8% predicting the direction of the stock's price in relation to a given index; with a backpropagation neural network, the precision was 83.4%. The main problem with this study is that the experiment is not reproducible, because the attributes that were used to create the trading rules were not specified, and neither was the fitness function with which they were evolved. It is likely that this information was kept private due to the authors' affiliation with a capital management company. For obvious reasons, AI research done at investment companies is usually considered a trade secret, so it is possible that a lot of valuable research on this topic will never see the light of day. Regardless, there is currently no lack of open information on the subject, with numerous publications and books focusing on the theme (Kovalerchuk & Vityaev, 2000).

After going through much of this literature, it is our opinion that there is a general consensus in favour of the usefulness of data mining models for performing financial forecasting – most articles report positive results, and in doing so provide empirical evidence against the efficient market hypothesis. However, this consensus could be due to the fact that positive results make for better publications, and there is a bias against negative studies (Dickersin, 1990), or because researchers are less likely to submit negative or inconclusive results – the well-known file drawer effect (Rosenthal, 1979); also, the (deceiving) positive results reported in some studies can be directly attributed to poor data mining technic. In spite of this, we agree that data mining should be well-suited for the task of modelling financial data; the fact that several investment companies are currently using data mining models as part of their trading strategies is, in itself, a good endorsement of the models' utility in the field of finance.

2.2 Forecasting with Ensembles and Hybrid Systems

The articles referenced in the previous section demonstrated, to a certain degree, that simple data mining models might be able to output profitable financial predictions. It is conceivable that more complex AI-based forecasting systems can yield even better results. Abraham *et al.* (2003) reached that conclusion in a study that showed that a hybrid intelligent system could outperform standalone models. They compared the performance of a neuro-fuzzy system with that of an artificial neural network trained with the Levenberg-Marquardt algorithm, a difference boosting neural network and a support vector machine. These models performed a regression on the open, maximum and closing values of the NASDAQ_100 and the NIFTY stock indices, and were subsequently utilized to get one-day-ahead predictions for the values of these indices in out-of-sample data. The performance metrics considered were the root mean squared error (RMSE), the maximum absolute percentage error (MAP) and the mean absolute percentage error (MAPE). Their results indicate that the indices' value on any given day offer some insight into their closing value the following day. For either of them, none of the models had a RMSE over 0.03 or a MAPE above 10%; based on the MAPE metric, which measures, percentage-wise, how close the predictions were to the expected values, the neuro-fuzzy system outperformed all the models when predicting the NIFTY Index, and was a very close second to the support vector machine when doing so for the NASDAQ 100 Index. From what we can tell, the index data was not differenced prior to training the models. This might have improved the results – differencing financial time series is an important pre-processing step, because it makes the series stationary, and consequently removes stochastic trends (Franses, 1998). Non-stationary data does not exhibit a tendency for mean reversion; its statistical properties (like the mean and the standard deviation) vary with time, ergo it is more difficult to predict, and often impossible to model. A study by Qi and Zhang (2008) demonstrates the importance of differencing the data prior to training the artificial neural networks; using data generated by six different processes, they tested the models with the original and the differenced data, and verified that the latter considerably decreased the networks' RMSE and MAE. Even without differencing, the results obtained by Abraham *et al.* were relatively good, and do seem to reinforce the idea that a more complex system will outperform standalone models. In another study, Abraham (2005) compared a neuro-fuzzy system, a neural network trained with the scaled conjugate gradient algorithm, a CART decision tree, a multivariate adaptive regression splines model (MARS) and a hybrid CART-MARS model. This time around, the models were used to predict the exchange rate of five currency pairs, with a time frame of one month. The predictions were made using time windowing, i.e., the most recent monthly exchange rates were utilized as inputs to the models, which tried to predict the exchange rate in the following month. The final results included the RMSE of the models' predictions, and charts where those predictions were plotted together with the desired values. This type of comparison chart is common in financial forecasting studies; however, this is not a good way to report the results, because these charts can be quite deceiving – even if they show that the forecasted values are very close to the desired values, the forecasts might still be useless. Swingler (1996) demonstrated this pitfall with the following example: if we create a model that always predicts that the next value in the series will be the same as the current value (an optimum strategy according to the martingale hypothesis), and juxtapose its predictions with the series being predicted, the forecasted values will seem very accurate to the naked eye; yet, if we examine the chart closely, we will see that the predicted values are lagged, which means they are worthless in practice. Consider the example presented in Figure 2; although it looks like the predictions accurately track the expected values, we would not be able to use these predictions to trade profitably, because they are always one step behind. This goes to show just how important it is to pick good performance metrics, and to analyse the results thoroughly, when researching this topic. The RMSE values in Abraham's study allow for a much better comparison between the models; they show that the two hybrid systems outperformed all the standalone models, when tested with four of the five currency pairs. Once again, the more complex solutions yielded the best results. Another interesting hybrid approach to the financial prediction problem is described by Yu *et al.* (2005a); their system combined an expert system with an artificial neural network trained with the backpropagation algorithm; the neural network was responsible for predicting future exchange rates, while the expert system was responsible for combining those

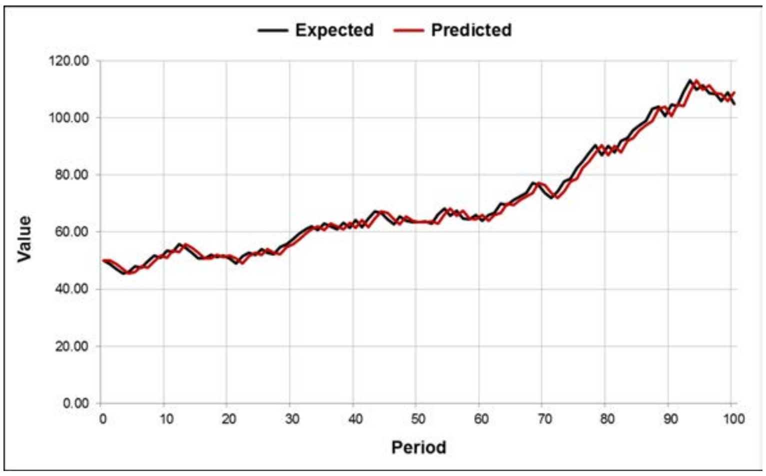


Figure 2. Misleading results: predicting that the next value will be the same as the current value.

predictions with expert knowledge to make trading suggestions. Their simulated trading results indicate a return of between 13% and 15% per year after commissions, depending on the currency pair. On their own, the returns of the neural network and the expert system were considerably lower, which confirms that the hybrid solution was better than the simpler systems. Sadly, their experiment is not reproducible, because the inputs to the neural network and the rules in the expert system were not provided.

Some researchers addressed the financial prediction problem with more “creative” strategies. Sehgal and Song (2007) developed a classifier-based system that performed text mining on user postings in message boards, and then used this information to do daily predictions regarding the direction of the price of several stocks. This is a strange approach – anyone who has ever visited one of these boards is well aware that most posters are biased, due to their own stock positions, and that the vast majority of the posts do not contain any useful information. Nonetheless, the researchers claimed they found a relationship between what they called the “web sentiment”, and the daily price of several individual stocks; according to their results, if the text mining mechanism responsible for measuring the sentiment was coupled with a decision tree, price direction predictions could be made with very high accuracy (above 80% for Apple’s stock, for example). These results are a bit implausible, not only because of the study’s premise, but also because they imply that the stock market is extremely inefficient. Since no information was provided regarding how the tests were done, no further analysis of the results is possible; one thing is certain: it would be virtually impossible for their system to maintain that level of accuracy in the long run. Mittermayer’s (2004) strategy to apply text mining in stock price prediction was more sensible. He created a system called NewsCATS, which forecasted the movement of a company’s stock price based on press releases posted online: as soon as a press release was published, the system used a support vector machine to categorize it as either “good news”, “bad news” or “no movers”; the “good news” were those expected to cause an average price increase of at least 1% during the 60 minutes after the release, and a minimum increase of 3% at any point during that period; the "bad news" were those expected to cause a price drop of at least 3% during the next 60 minutes, and an average decline of at least 1%. By implementing a trading strategy that bought the stock when "good news" were released, and shorted the stock when "bad news" were posted, he achieved an average profit per trade of up to 0.21%. While this is a good performance, it is not very realistic: it does not take into account the trading commissions, or the slippage costs that are often associated with short term news trading systems (due to the higher volatility). Also, the profit was calculated using an optimized take-profit target that was chosen after the trades were simulated; this optimized target worked well for the simulated trades, but there is no guarantee that it would also work well with future trades – thus, the results are biased, because information that was only available after the trading simulation had ended was utilized to optimize the investment strategy. To his credit, Mittermayer also reported the return that would be obtained without the take-profit orders; this was considerably lower, at just 0.11% per trade, which might not be sufficient to compensate for the trading costs. Nevertheless, his research demonstrates that it may be possible to create a profitable trading system with text mining, by continuously monitoring and classifying press releases. Unlike Sehgal and Song's idea, this strategy actually makes intuitive sense.

Instead of mining text, Li and Tsang's (1999) resorted to the more traditional approach of using historical prices to derive financial forecasts – the technical analysts' way. Their strategy was to utilize a genetic algorithm to evolve decision trees. The initial population consisted of a set of random decision trees tasked with predicting if the Dow Jones Industrial Average Index was going to increase at least 2.2% in the next 21 trading days; the training attributes were simple technical indicators, like moving averages. The genetic algorithm was used to obtain better trees, by optimizing their precision; after repeating the experiment 10 times, the average accuracy of the best decision trees in the final populations was 54.78%. Several standalone C4.5 decision trees were also trained, using different parameters; they underperformed the genetic decision trees, with an average accuracy of 53.40%. Choudhry and Garg (2008) also employed a genetic algorithm as part of a hybrid system. However, in their experiment, the objective of the algorithm was to optimize the set of training attributes for a support vector machine, which was going to output one-day-ahead price direction predictions for several stocks; the list of potential attributes was very large, and included, among other things, several technical indicators, and the prices of correlated stocks. The average accuracy of the hybrid system, using test data for three stocks, was 60.5%; this was considerably better than the 56.8% accuracy of a support vector machine that was trained with all the attributes. This difference is not surprising, because support vector machines are sensitive to the presence of irrelevant attributes in the training data; so, it makes sense that picking the best set of training attributes with the genetic algorithm will improve their performance. This attribute selection mechanism would not be necessary, if we were dealing with a data mining model capable of doing its own selection during training (like the C4.5 decision tree). A similar study by Yu *et al.* (2005b) describes the exact same hybrid mechanism, consisting of a genetic algorithm for attribute selection, and a support vector machine for forecasting. They used it to make one-day-ahead predictions for the direction of the S&P 500 Index; with the equivalent to one year of out-of-sample data, their hybrid system achieved an accuracy of 84.6%; this compares with an accuracy of 56.1% for an ARIMA model, 69.8% for a backpropagation artificial neural network, and 78.7% for a standalone support vector machine. While we are a bit sceptical about these accuracy numbers, their results at least confirm that a hybrid solution can outperform simpler mechanisms, and reinforce the importance of proper attribute selection in data mining. Kwon and Moon (2004) also utilized a genetic algorithm, only this time with the intent of creating an ensemble of recurrent neural networks. Starting with a population of random networks, the genetic algorithm was used to optimize the weights of their synapses, and an ensemble with the best recurrent neural networks was created from a subset of the final population. This ensemble was tested with data for various stocks, for which it had to predict the one-day-ahead price direction. The results obtained led the authors to conclude that their system could produce better results than the buy-and-hold investment strategy.

Grosan and Abraham (2006) applied genetic programming to the modelling of financial data; this evolutionary method is similar to genetic algorithms, only it is meant for evolving computer programs. Their system consisted of an ensemble of genetic programming models; the objective of the programs optimized by these models was to do daily predictions for the values of the NASDAQ_100 and the NIFTY indices. An artificial neural network and a neuro-fuzzy system were also evaluated, for comparison purposes, using several metrics. According to the MAPE, the hybrid model and the ensemble performed better than the standalone neural network: with the NASDAQ_100 Index data, the neuro-fuzzy system got the best result (a MAPE of 7.6%); with the NIFTY Index data, the best performing was the ensemble (a MAPE of 2.8%); when used separately, the genetic programming models underperformed the ensemble. So, once again, the more complex systems got the best performances. Chou *et al.* (1996) also tested an ensemble. They created a decision support system consisting of a combination of rule-based artificial neural networks, and used it to trade the Taiwan Stock Exchange Weighted Price Index; inputs to the neural networks included not only historical values, but also technical analysis indicators, such as moving averages and the RSI. Their system attempted to predict the index's short and long term trends, and these predictions were utilized to open trades; in their experiment, the system did not open many trades, but still got incredible results: an average annual return of 44% in 4 years' worth of test data. Regrettably, this experiment is not reproducible, because the authors did not provide any information regarding the settings of the system; thus, we cannot be certain that these results were not tainted by the excessive "tweaking" of the settings.

The articles mentioned in this section described two possible strategies for implementing financial forecasting mechanisms: using a hybrid system (i.e., a combination of different artificial intelligence techniques) or using an ensemble (i.e., a set of data mining models whose predictions are aggregated into a single forecast). Overall, the articles' results demonstrated that these mechanisms will generally make better financial predictors than standalone data mining models. This is an important conclusion, which we will take into consideration when we start developing our own financial prediction system.

2.3 Trading with Autonomous Agents

So far, we have analysed several data mining models and systems that could be used to help human traders reach trading decisions. The main goal of our work is to take the next logical step forward, and research intelligent agents that can actually replace these traders. In order to supersede its human counterpart, an intelligent trading agent will need to be capable of trading autonomously; as we see it, this implies meeting the following requirements:

- It should be able to decide when to buy or short sell a given financial instrument, and when to close the trades;
- it should be capable of performing money and risk management;
- it should be able to keep learning over time, even as it trades;
- it should be capable of adapting to changes in market conditions; in particular, it should be "intelligent" enough to stop trading when the market becomes less predictable, and to resume trading when conditions improve.

Unfortunately, there are no comprehensive studies focusing on the development of this type of agent. Most literature about trading agents does not actually refer to agents that negotiate financial instruments, but rather to agents that participate in auctions. Many articles on this topic have come out of two popular trading agent competitions, the TAC Classic (Wellman *et al.*, 2001) and the TAC SCM (Sadeh *et al.*, 2003). An agent in the TAC Classic acts as a travel agency, competing with other agents in simultaneous auctions to assemble travel packages for eight costumers with specific requirements. TAC SCM, on the other hand, is a supply chain management competition, in which each agent acts as a PC manufacturer and competes with other agents for components and customer orders. The goals and strategies employed by these trading agents are completely different from those of a financial trader, so the literature available on these competitions does not relate to our research. Another interesting topic that does not directly apply is that of agent-based computational finance, for which there are also many studies available (Tsang & Jaramillo, 2004; LeBaron, 2006); these studies focus on artificial financial markets (i.e., virtual markets composed of only the software agents being tested), rather than real financial markets; their aim is to model the way the trading agents reach equilibrium in the fabricated market (Palmer *et al.*, 1999; Chen & Yeh, 2001). The final objective here is to understand the behaviour of the markets, as it emerges from the interactions between agents, in order to explain certain features that financial time series usually exhibit, such as conditional heteroskedasticity (i.e., clustered volatility, or the tendency for outlier observations to emerge in clusters), large kurtosis (i.e., fat tails, or the tendency for outlier returns to occur more often than expected), mean reversion (i.e., regression towards the mean after extreme moves), and the cycles of bubbles and crashes (Hommes, 2006). Studies on this topic might eventually prove useful for testing the impact of new economic policies or rules in virtual markets, prior to their introduction in real markets (Buchanan, 2009; Farmer & Foley, 2009). This objective is clearly different from what we are trying to achieve: in our research, we will not be modelling financial markets, but rather the behaviour of the traders participating in them; also, our focus will be on the real financial markets, because we want to create agents that are useful in practice. Although rarer, there are a few studies stating similar goals. Schulenburg and Ross (2000), for example, described the implementation of trading agents as learning classifier systems, or LCSs (Booker *et al.*, 1989); in a LCS, a population of weighted classification rules is evolved using a genetic algorithm. The most interesting thing about this study is that it addresses the issue that, since the agents will be inserted in a perpetually changing noisy environment, they need to be able to keep learning as time goes by, so that they can adapt to new market conditions. The learning is performed by the genetic algorithm, which allows the population of trading rules to be continuously improved by crossover (combining the best rules) and mutation (slightly changing existing rules). After testing the agents, the researchers reported that their profit was slightly better than that of the buy-and-hold strategy. Since the test period was relatively short, we cannot predict how well these agents would fare going forward; the fact that they were configured to trade only on the long side (i.e., no short trading was allowed) is a disadvantage, because in the real markets the prices may keep falling for extended periods of time. In another study, Lee (2004) suggested using the iJADE multipurpose framework to create an intelligent stock trading agent that made price predictions using a recurrent radial basis function network; the results of this study are presented in a format (RMSE and average percentage error) that does not allow us to draw any conclusions regarding the agent's potential for real life trading; also, the study does not address the agent's need to adjust to different market conditions, because its architecture does not allow it to learn.

Luo *et al.* (2002) proposed a multi-agent system to be used as an aid for human traders; each agent in this system is responsible for a specific task (technical analysis, fundamental analysis, risk management, etc.). Lee *et al.* (2007) followed a similar approach; they presented a multi-agent system consisting of 4 different agents: the buy signal agent, the sell signal agent, the buy order agent and the sell order agent; it is unclear why these functionalities had to be assigned to different agents. These multi-agent systems, intended as simple suggestion mechanisms, are out of the scope of our work, because our focus is on truly autonomous agents. Castro and Sichman (2009) described some agents that fit this description, in an article where they also proposed an open source financial market simulation tool; however, their emphasis was on the simulation tool, not the trading agents, which ended up being extremely naïve. It may be argued that these agents, as well as some of the other agents that were previously described, are not true intelligent agents. The same can be said of most of the agents that participated in the Penn-Lehman Automated Trading Project (Kearns & Ortiz, 2003) competitions. These entities are just simple trading bots that open and close trades automatically, but cannot learn new patterns, or adjust to changes in the environment. They partially meet our autonomy requirements, in that they are capable of deciding when to enter and exit the markets. But ultimately, these bots are nothing more than hard-coded automated trading strategies of variable complexity. While they can open and close trades on their own, they are not really autonomous, because they need to be closely monitored at all times – since the bots cannot adapt to the market as time progresses, one of two things will happen when their strategies becomes outdated: either they are manually stopped and retrained with new data, or they will continue to trade until they go bankrupt. Unlike these simple trading mechanisms, we intend to develop agents that completely fulfil all the autonomy requirements that were previously listed, so that they may be left to trade without human supervision for an indefinite period of time.

It is fair to assume that the most interesting studies on this topic are being done by financial companies that engage in algorithmic trading; understandably, these studies will never be made public. Furthermore, even the returns of their automated systems are somewhat private, so it is hard to quantify what one can reasonably expect from them in the long run. This lack of clarity is what motivates our work: an in-depth study on the design and performance of intelligent trading agents and agent-based investment systems. While pursuing this work, we will try to avoid all the problems and pitfalls that were detected in some of the articles referenced in this chapter; that is to say, we will make sure that the following conditions are fulfilled:

- the experiments must be reproducible, i.e., all the implementation details must be provided (that is one of the main principles of the scientific method);
- the test data used for evaluating the agents needs to be comprehensive enough, i.e., it must encompass a large period of time, and must include periods of extreme volatility; this requirement is particularly important because quantitative strategies tend to fail during these periods (Khandani & Lo, 2007); also, the test data should include periods

when the instruments' prices are trending upward and periods when they are trending downward, to ensure the agents can operate properly in both situations;

- biased results must be avoided at all cost; this implies keeping the training data completely separated from the test data, never using test data results for optimization purposes, and always reporting the results of all the tests made (i.e., the worst performing agents must not be discarded, to avoid survivorship bias); ideally, a forward-testing mechanism should be provided to complement the backtesting results, in order to eliminate doubts that these could be biased;
- the performance metrics utilized in the evaluation of the agents must suit the problem at hand, i.e., the emphasis should be put on metrics deemed important by financial traders (such as the return and the maximum drawdown), in lieu of metrics like the accuracy or the RMSE.

We believe these requirements are all of vital importance, and should serve as a guideline for any new study on this subject.

Since this type of research has obvious practical ramifications, it makes sense to aim it towards the objectives of those who might eventually put it to use. For this reason, we will be implementing the agents according to what we consider to be the necessities and goals of the financial community. These goals will be presented in the next chapter, along with our proposal for a new agent architecture, which is intended for the development of trading agents that pursue those objectives.

Chapter 3

A Hybrid Cognitive Architecture for Intelligent Trading Agents

As stated by Jennings and Wooldridge (1998), one situation in which it is reasonable to apply agent technology is when an intelligent agent is an appropriate metaphor for a given functionality. We believe there is a clear-cut example of one such metaphor in the world of finance. Consider the inner workings of a hedge fund, i.e., a loosely regulated private investment firm that trades other people's money for a fee, and is allowed to buy and short sell a wide range of financial instruments. The typical hedge fund employs several traders, each being responsible for negotiating a specific set of financial instruments to try to get the best return possible. These traders likely cooperate with each other, in order to maximize the profit of the hedge fund as a whole. Clearly, a multi-agent system is a natural metaphor for this type of organization, with the intelligent trading agents playing the part of the human traders. We should note that, with this metaphor, we are not implying that being a successful financial trader is in any way correlated to being intelligent. In fact, we intend to demonstrate in this chapter that profitable trading can be achieved by chance alone. Profitability aside, the software trading agents in our hedge fund scenario may be considered intelligent in the sense that they act rationally, i.e., they exhibit autonomous goal-oriented behaviour.

We mentioned previously that well-programmed intelligent trading agents should offer significant advantages over human traders, among which being much cheaper (no salary or annual bonus), being able to trade 24 hours a day (no breaks or vacation time), being emotionless and easier to manage (no rogue traders), and being able to trade much faster – if a trading strategy is based solely on number crunching, and the intelligent agents have access to powerful hardware and fast network connectivity, they will always outperform their human counterparts. In view of all these advantages, we think this subject is worthy of a thorough exploratory study. Our research will begin with the design of a novel agent architecture, which will be used as the basis for the financial trading agents. Numerous generic architectures have already been proposed in AI literature, several of which were described in Section 1.2. However, none of them targets the financial field specifically. Cognitive architectures are, for the most part, more concerned with general human-like behaviour than with particular competencies, i.e., they attempt to address cognition as a whole rather than the cognitive behaviour associated with specific tasks. Take the SOAR architecture (Laird *et al.*, 1987), for example. Using it in the development of trading agents would require that all trading knowledge, as well as the actual skill behind successful trading, could be expressed as rules and facts. In the highly competitive and ever changing environment that characterises financial markets, we hardly believe that is possible. The same problem would arise if we tried to adapt other symbolic cognitive architectures – such as the PRS (Georgeff & Lansky, 1987), the ACT-R (Anderson, 1996) or the 4CAPS (Just & Varma, 2007) – to the development of trading agents with any chance of being successful in the long term. PRS-based BDI agents would be particularly inadequate for this task, because they lack the ability to learn, and their planning algorithm is not well suited for an environment where conditions might change dramatically at any second. Because of these limitations, a symbolic cognitive architecture is likely not the best option for what we are trying to accomplish. On the other hand, a hybrid cognitive architecture (i.e., one that combines the symbolic and connectionist stances) might be exactly what we are looking for.

Unfortunately, the ones currently available, like the LIDA (Friedlander & Franklin, 2008), the CHREST (Gobet *et al.*, 2001), the DUAL (Nestor & Kokinov, 2004) or the CLARION (Sun *et al.*, 2001), are meant for more general tasks, and do not fit our objectives very well. Besides cognitive architectures, there are also a few behaviour-based architectures that we could consider using, such as Brooks' subsumption architecture (1990) or the GRL (Horswill, 2000). They are better suited for creating agents that execute concrete tasks, because they were devised with robotic agents in mind. However, they are best applied in the development of agents that are mostly reactive. Since we want our trading agents to be more proactive than reactive, this type of architecture is also not what we are after. Duch *et al.* (2008) published a comprehensive study summarizing the inner workings of the various agent architectures that were referenced so far, as well as several others. Going through their list, we can verify that many of these architectures were inspired by neurological and psychological studies, and attempt to emulate the way the human brain works (although in a very crude manner); hence, they aim to solve the problems of artificial consciousness and general intelligence. This is far different from what we are trying to do – we are not looking to determine how successful human traders think (arguably, an unattainable goal in the foreseeable future), we just want to devise agents that mimic their actions. Like Brooks, we put the emphasis on functionality, and we aspire to fit the theory to the domain, not "force" it. Creating our own agent architecture, rather than using one of the many readily available, gives us the flexibility to fully customize it for the intended task. By having the architecture address the many specificities and quirks that characterise financial trading, we are hopeful the agents' "intelligence" will end up emerging from their actions. In this chapter, we will be describing this new hybrid cognitive architecture, which is meant specifically for the development of autonomous agents that trade financial instruments. This architecture is composed of three modules:

- the prediction module, responsible for forecasting the direction of the price of a financial instrument;

- the empirical knowledge module, responsible for deciding how much to invest in each trade;
- the domain knowledge module, responsible for incorporating expert knowledge into the trading decisions, such as the timing for closing open trades.

The contribution of each module to the trading performance will be demonstrated with the step-by-step implementation of two intelligent agents. One will be used to trade the USD/JPY currency pair with a time frame of 6 hours, while the other will be day trading the ADBE stock. In the last section of this chapter, we will be presenting a multipurpose software shell that implements the proposed architecture, and allows the rapid development of intelligent agents that can trade any type of financial instrument.

3.1 Goals of Intelligent Trading Agents

In order to trade a financial instrument without supervision, a software agent will need to be able to make several decisions on its own. More concretely, it will have to be capable of answering questions like:

- When should a financial instrument be bought or short sold?
- How much should be invested in each trade?
- When should an open trade be closed?

The way the agent answers these questions will depend on what it is trying to accomplish. Since our objective is to create agents that mimic the activity of typical hedge fund traders, their goal should be to try to obtain the maximum profit possible, while simultaneously minimizing the risk. We can measure how well an agent achieves this objective with two metrics: the return on investment and the maximum drawdown. The return on investment is the percentage ratio between the agent's profit and the investment capital it started with. The maximum drawdown, on the other hand, measures the historical maximum peak-to-valley decline in its equity, i.e., the maximum accumulated loss the agent experienced while trading. Figure 3 shows the values for these two metrics, considering a scenario in which a trader bought $100,000 of ADBE stock in February of 2006, and held it for 3.3 years. According to the equity curve in this figure, the $100,000 initial investment would turn into around $64,000 in the end, which corresponds to a return of -36.0%. The maximum accumulated loss occurred between November of 2007 and March of 2009; at the start of this period, the trader's account balance was around $118,500, and at the end it was $39,500, so the maximum drawdown is 66.7%. This metric is important because it measures how risky the trader's strategy was in the past. The large maximum drawdown in this example proves that the buy-and-hold strategy is extremely unsafe at times: had the trader purchased the stock in November of 2007, the buy-and-hold strategy would have yielded a massive loss of 66.7% after just one and a half years. Even worse, if the purchase was made using leverage, this loss would have originated a margin call, wiping out the trader's account. This scenario demonstrates why the maximum drawdown is an important metric for the risk associated with a trading strategy. That is, of course, assuming that the measure of how dangerous a strategy was in the past can offer some insight on how it might perform in the future. This is a dangerous assumption, because there is no guarantee that a larger accumulated loss will not occur later on.

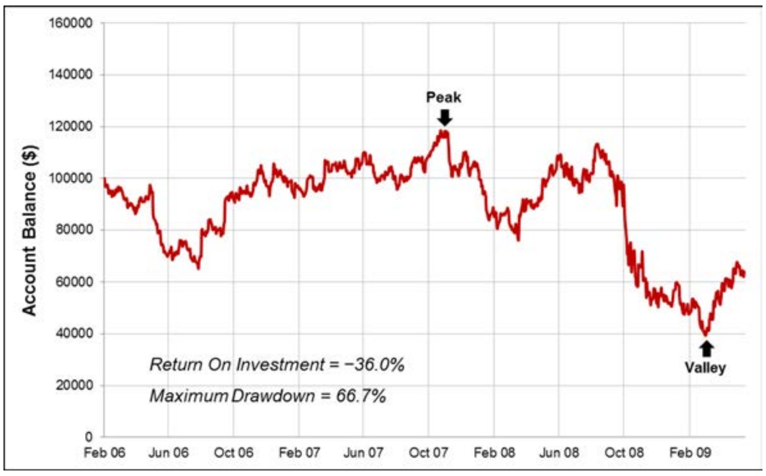


Figure 3. Return and maximum drawdown metrics of a buy-and-hold strategy using the ADBE stock.

Likewise, a strategy's return in the past is just a rough estimate of how much profit it might yield in the future, not a certainty. Investment companies are legally obliged to acknowledge the unreliability of these two metrics by including a disclaimer in their performance reports stating that "past performance is not a guarantee of future returns". Nevertheless, the only way to evaluate the success and potential of these entities is to look at their historical trading track records, so the return and maximum drawdown metrics are unavoidable. Instead of measuring the agents' performance directly with these two gauges, we will be using two related metrics. The first is the ratio between the total return and the maximum drawdown, which we named "RMD ratio":

$$RMD\ ratio = \frac{Return\ Since\ Inception}{Maximum\ Drawdown\ Since\ Inception}$$

This is a pain-to-gain ratio similar to the Calmar (Young, 1991) and the MAR ratios, which are frequently used by retail investors to compare the performances of different investment funds. This ratio allows us to measure the risk-adjusted performance of a trading strategy (in the past). The higher the RMD ratio, the bigger the strategy's return was compared to its maximum drawdown; strategies with higher RMD ratios are theoretically better suited for trading with leverage because, assuming they will maintain the same level of performance going forward, increasing the leverage will increase the return much more than it will increase future drawdowns. The other metric we will use in our study is the return per trade, which is calculated by dividing the total return on investment by the total number of trades. A high return per trade is extremely important, because there are significant costs associated with real life trading. These costs include, among other things, commissions, spreads and slippage. If the return obtained in each trade is not big enough to at least make up for its cost, the trader will end up losing money. It is not uncommon for strategies that seem very profitable on paper to fail miserably, once these costs are taken into consideration.

In addition to the return and the maximum drawdown, financial players often use metrics like the Sharpe (Sharpe, 1966) and the Sortino ratios (Sortino & Price, 1994) to compare investment performances. These metrics imply the existence of a risk-free investment (usually U.S. Treasuries), and measure the return of an investment strategy in comparison with that risk-free return. This is reasonable, because there is no point in engaging in risky investments when a similar return is achievable without incurring any risk. The reason why we decided not to utilize these ratios in our research is that we do not believe there is such a thing as a risk-free investment. Even if some government debt seems extremely safe right now, there is no telling how things might change in the future – history has shown that, every now and then, a black swan event (Taleb, 2007) will sneak up on investors and completely discredit the investment dogmas *du jour*, so today's riskless trade might be tomorrow's investment nightmare. Also, in our opinion, those ratios obfuscate the results. The absolute return and the maximum drawdown are clearer: they are easier to interpret, and make it simpler to decide if the profit of an investment strategy in the past was worth it, considering its risk.

Now that we have made our case as to how the trading agents should be evaluated, we can define their goals accordingly: they should attempt to maximize the RMD ratio of their trading strategy (meaning, they should try to obtain the best return possible while keeping the drawdowns relatively low), and also maximize the return per trade (so that their profit is not completely wasted on commissions). In the next sections we will present an agent architecture that was created with these objectives in mind.

3.2 Predicting the Direction of the Price Using Data Mining

Just like the human entities they attempt to mimic, software trading agents must decide when to buy or short sell a financial instrument. Numerous studies have indicated that data mining models may be used for this purpose, several of which were mentioned in Chapter 2. We will be following a similar strategy, by incorporating a model-based prediction mechanism in the agents' architecture. Obviously, this mechanism can be implemented using a multitude of strategies (Barbosa & Belo, 2009a). The simplest approach is to train a data mining model to predict the direction of the price of the financial instrument that we want to trade; the model's predictions can then be utilized to decide if the instrument should be bought or sold short. In order to demonstrate this procedure, we trained seven different models using historical data relative to the USD/JPY exchange rate. Details on the training attributes and parameters will be discussed in later chapters, along with the reasoning behind each configuration setting; for now, we will be focusing solely on the models' performance. The USD/JPY raw data was segmented into instances, each corresponding to a period of 6 hours. Five of the models were trained to classify the instances, in order to predict the direction of the price of the currency pair in subsequent 6-hour periods. These predictions corresponded to one of two classes: "the price of the USD/JPY pair will increase in the next 6 hours" (class *UP*) or "the price of the USD/JPY pair will decrease in the next 6 hours" (class *DOWN*). The other two data mining models were trained for regression with the same data; instead of predicting the direction of the price, they forecasted the pair's price change (in percentage) in the next 6-hour periods, and these numeric forecasts were converted into one of the classes: if the model predicted a negative price change, its class prediction was *DOWN*, otherwise it was *UP*. Each of the seven data mining models was utilized to implement a simple trading bot, according to the architecture shown in Figure 4. Notice we are using the term "trading bot" instead of "intelligent trading agent", because the entities implemented with this architecture cannot be considered intelligent. While they can act autonomously, they are nothing but simple hardcoded programs that lack the ability to adapt to changes in market conditions. Their behaviour is also not guided by the objectives that were defined for our intelligent agents: to maximize the return while attempting to minimize the risk. The pseudocode describing the implementation of this architecture is listed in Algorithm 1, and the corresponding UML sequence diagram is displayed in Figure 5.

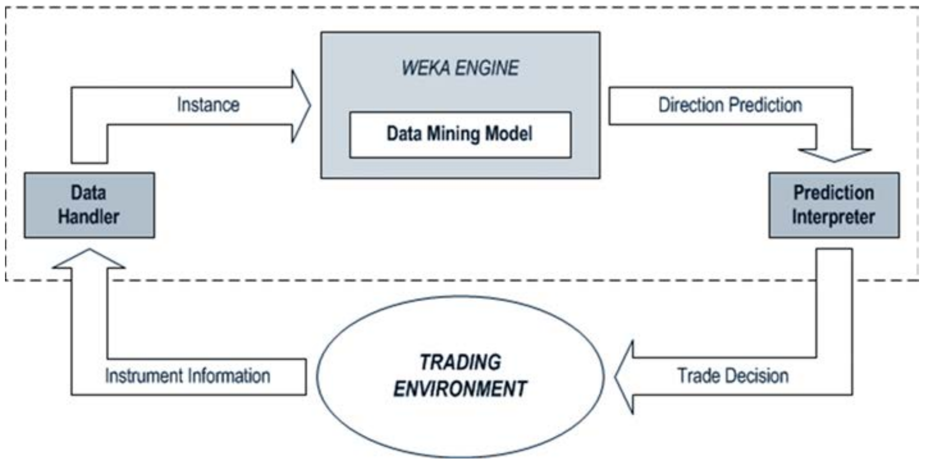


Figure 4. Trading bot architecture.

```
Algorithm TradingBot
Inputs:
    ticker          // instrument to trade
    amount          // amount to invest in each trade
    model           // data mining model that will do the predictions

BEGIN
    Repeat
        confirmation ← wait_for_end(period)                // wait for current trading period to end
        confirmation ← close_trade(tradeID)                // close open trade
        periodData ← get_financial_data(ticker,period)     // get instrument's financial data for the period
        confirmation ← add_to_data(ticker,periodData)      // add period data to the database
        historicalData ← get_data(ticker)                  // get the instrument's historical financial data from the database
        instance ← create_instance(historicalData,model)   // create instance for the period using the raw financial data
        class ← classify_instance(instance,model)          // predict price direction in new period by classifying the instance
        If class = UP Then
            tradeID ← buy_instrument(ticker,amount)        // buy if class predicted is UP
        Else
            tradeID ← short_instrument(ticker,amount)      // short sell if class predicted is DOWN
        EndIf
    EndRepeat
END
```

Algorithm 1. Trading bot pseudocode.

The data mining models were trained using the Weka API[8] (Witten & Frank, 2005), each with a specific set of attributes. The training instances were extracted from historical data corresponding to the period between May of 2003 and December of 2006, for a total of around

[8] The Weka API is available at http://www.cs.waikato.ac.nz/ml/weka/.

4,000 instances; the 50 instances that make up the first 2.5 weeks of January of 2007 were used to test the models; the subsequent 2,510 instances (up to the middle of May of 2009) were reserved for out-of-sample performance evaluation. These time periods are delimited in Figure 6, which shows the USD/JPY exchange rate since 1975. There are two things worth noting in this figure: first, it is clear that the prices in the training data are considerably less volatile than in other periods in the past; second, the price changes in the out-of-sample data look quite different from those in the training data, and that may hurt the models' accuracy.

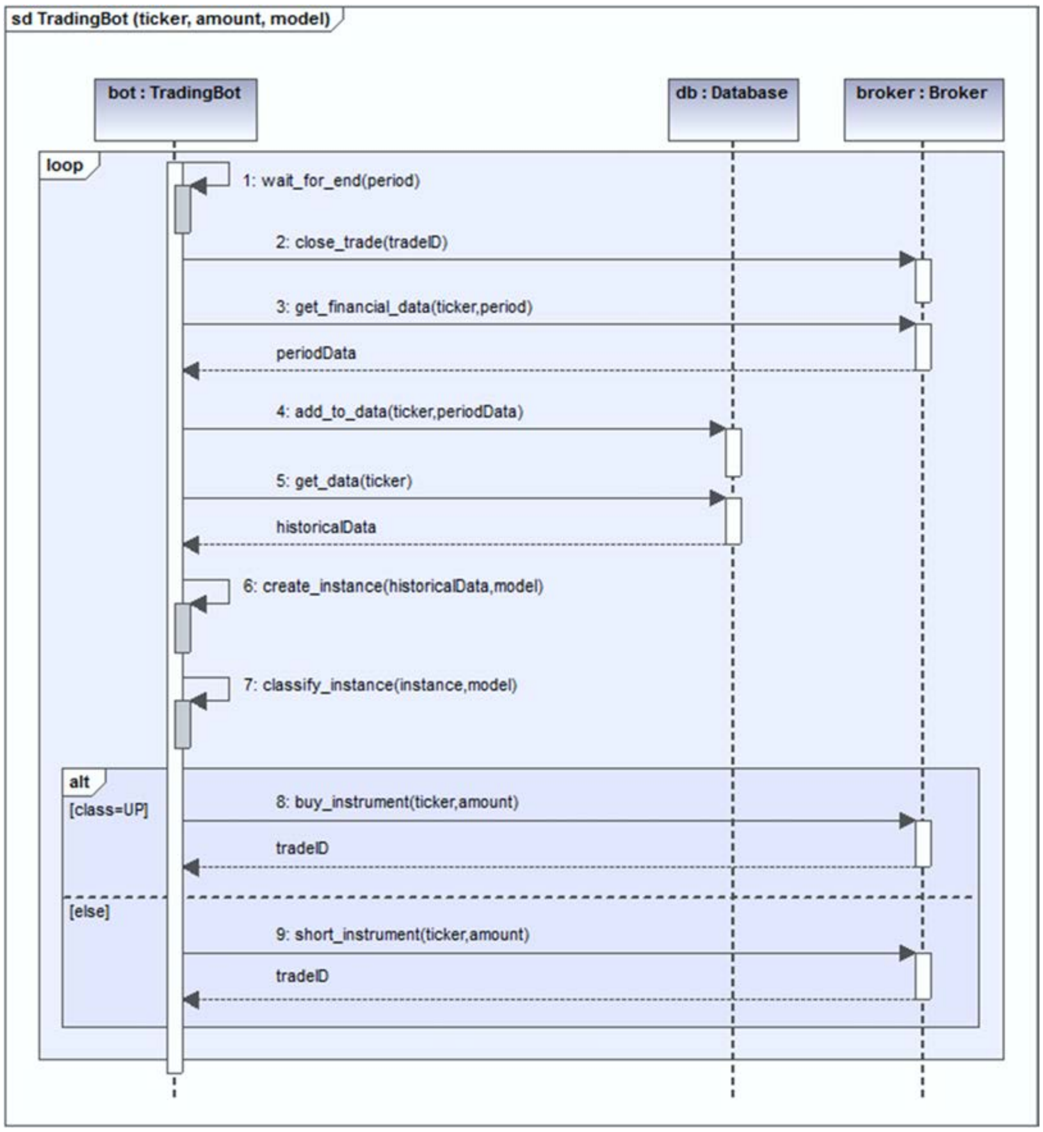

sd TradingBot (ticker, amount, model)
bot : TradingBot
db : Database
broker : Broker
loop
1: wait_for_end(period)
2: close_trade(tradeID)
3: get_financial_data(ticker,period)
periodData
4: add_to_data(ticker,periodData)
5: get_data(ticker)
historicalData
6: create_instance(historicalData,model)
7: classify_instance(instance,model)
alt
[class=UP]
8: buy_instrument(ticker,amount)
tradeID
[else]
9: short_instrument(ticker,amount)
tradeID

Figure 5. Trading bot UML sequence diagram.

Once the implementation of the seven bots was completed, each of them made predictions and simulated trades for the 2,510 out-of-sample instances. As previously mentioned, each instance corresponds to a 6-hour period, thus our trading simulation implies that the bots would be opening a new trade every 6 hours, starting on Sunday at 18:00 GMT till Saturday at 00:00 GMT. For each instance, if the bot's data mining model predicted a price increase, a long trade was simulated; if it predicted a price decrease, a short trade was simulated. If the forecast was accurate, the absolute value of the USD/JPY percentage price change in the corresponding period was added to the bot's return, otherwise it was subtracted. Notice that this method of calculating the profit implies that the bots were using a fixed trade size throughout the simulation. Figure 7 shows the cumulative returns that the five bots that used classification models obtained in the out-of-sample period; the cumulative returns of the two bots that used regression models are shown in Figure 8; the final statistics are summarized in Table 1. We must note that these are gross returns, i.e., they do not account for the trading costs.

When analysing the bots' performance, we were surprised to see that all of them were profitable at the end of the simulation period, a remarkable feat even if we consider that this profit is not net of expenses. Still, their trading results were nothing special. Several bots experienced

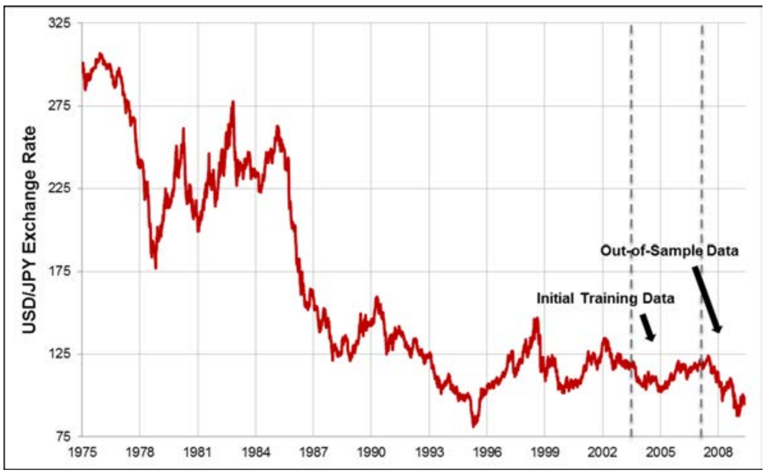


Figure 6. Historical USD/JPY prices.

large drawdowns, which took them several months to recover from. The worst offenders in this regard were the ones with the K*, the C4.5 decision tree and the Naïve Bayes classification models. The charts reveal that it took them a long time to reclaim historical peaks in their cumulative return curves – in real life, it is doubtful we would have the patience to wait that long for them to recover their losses. This fault needs to be emphasised. Human traders would find it very hard to keep their jobs if they kept losing money for an extended period of time. This is, after all, an activity where instant gratification is the most important factor. Having worked in the industry, we

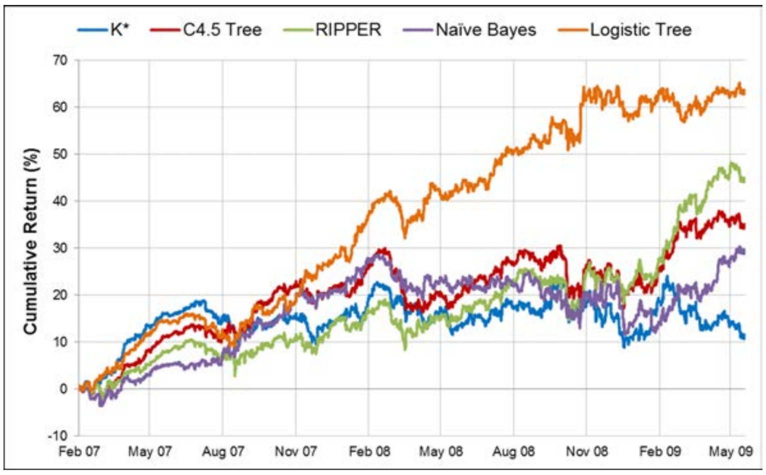


Figure 7. Gross cumulative returns of the USD/JPY trading bots with standalone classification models.

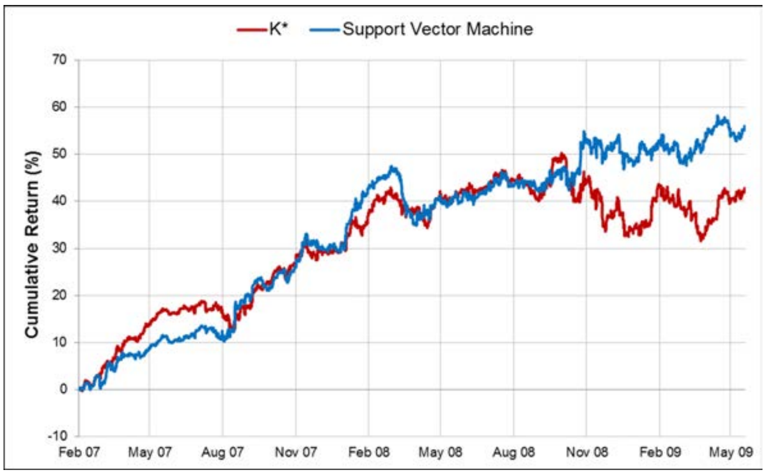


Figure 8. Gross cumulative returns of the USD/JPY trading bots with standalone regression models.

Table 1. Simulation results of the USD/JPY trading bots (excluding trading costs).

Model	Ret (%)	Max DD (%)	RMD Ratio	Ret/Trade (%)	Acc (%)	Trades
K* (classification)	11.6	14.8	0.79	0.0046	52.4	2,510
C4.5 Decision Tree	35.1	13.6	2.58	0.0140	53.2	2,510
RIPPER Rule Learner	45.0	11.0	4.08	0.0179	52.9	2,510
Naïve Bayes	29.7	17.1	1.74	0.0118	52.1	2,510
Logistic Model Tree	63.7	10.0	6.37	0.0254	54.3	2,510
K* (regression)	42.8	18.7	2.29	0.0170	52.9	2,510
Support Vector Machine	56.0	12.6	4.43	0.0223	53.4	2,510

can confirm that it is common practice for trading companies to hire numerous junior trades, with the intent of firing the ones that cannot produce results in the first few months. Our trading bots must face the same scrutiny, so we must analyse not only their final returns, but also their path to getting them. By these standards, looking at the charts in Figures 7 and 8, we believe that only the bots with the logistic tree, the RIPPER rule learner and the support vector machine achieved a good enough performance: their cumulative return curves trend upward very steadily, with prior highs being overcome relatively fast.

Based on the statistics in Table 1, we can conclude that none of the seven bots was capable of predicting the short term direction of the USD/JPY exchange rate with acceptable precision. This low accuracy was somewhat expectable. Exchange rates are extremely "noisy" and difficult to forecast, especially at shorter time frames like the bots' 6-hour periods. Furthermore, we saw in Figure 6 that the training data did not look similar to the test data, which could explain why the models had trouble applying the patterns discovered in the training phase to the new data. Fortunately, low accuracy is not necessarily a big problem, because successful trading is measured in terms of profitability, not accuracy. Higher precision does not always translate into bigger profits, as the bots' results show: the least accurate bot (which used a naïve Bayes classifier) was almost three times more profitable than the least profitable bot (which used a K* classifier). Similarly, the bot that used the RIPPER model was simultaneously more profitable and less accurate than the bot with the C4.5 decision tree.

By most measures, the worst performing bot in the group was the one predicting the direction of the USD/JPY exchange rate with a K* classifier. Its insignificant return per trade of 0.0046% means that, once the trading costs are taken into account, this bot's profit will more than likely turn into a considerable loss. The bot that used the logistic model tree, on the other hand, achieved impressive results. It was not only the most profitable and most accurate of all, but also the one with the lowest maximum drawdown. Its RMD ratio at the end of the simulation period was 6.37; this signifies that, if we configured it to employ leverage and repeated the experiment, the increase in its final return would be 6.37 times bigger than the increase in its maximum drawdown. On the flip side, this bot's return per trade was just 0.0254%, which should barely make up for the trading costs.

On balance, the bots' results indicate that a prediction mechanism based on a single data mining model is not good enough to be utilized in the implementation of intelligent trading agents, because picking a good model is only easy in hindsight. Moreover, even if we are lucky enough to pick a model that allows the bot to trade profitably for an extended period of time, that in no way guarantees that catastrophic losses will not occur later on. In fact, it is almost certain that the model's performance will eventually degrade. If we look at Figure 6, we can clearly see that the USD/JPY exchange rate has historically shown many patterns that are not visible in the training data; in particular, this data does not contain a crash in the pair's price like those that occurred in the '70s and the '80s. Since the training data is the only source of information that the model possesses, it will only recognize patterns found within it. Hence, if history were to repeat itself, and another crash occurred in the USD/JPY price, the model would more than likely not be able to recognize that pattern. In addition to missing these past patterns, the described "train once, use forever" prediction method also implies that the bot will not be able to learn new patterns that might occur in the future, no matter how many times they happen. This is yet another reason why its performance might worsen as time goes by. One way to overcome this problem would be to periodically retrain the data mining model with new data, as it becomes available. While this would enable the bot to keep learning over time, it could lead to a new problem. Many data mining models are naturally unstable, i.e., their learning algorithms may generate completely different models from relatively similar training sets. That is the case of C4.5 decision trees (Quinlan, 1993), for example. If we retrain one of these trees with more data, the resulting decision tree could be completely different from the original, potentially degrading the performance of the prediction mechanism. In this situation, the bot will keep opening trades based on the predictions of the inaccurate model, and eventually it will get a margin call. That is the biggest problem with using a single model for making forecasts: if the market dynamics change and the model's accuracy declines, the bot has no means to adapt to these changes, and will continue trading until it goes bankrupt. An implicit requirement for an intelligent trading agent is that it must at least try to survive extreme changes in market conditions, like those caused by black swan events. For this reason, the trading bots' prediction mechanism was deemed inadequate – we had to look for a more "intelligent" solution, one that would make the agents more resilient to those rare events.

A research-proven alternative to making predictions with a single data mining model is to use an ensemble of models. Several empirical studies, such as those by Sollich and Krogh (1996) or Opitz and Maclin (1999), have demonstrated that a committee of classifiers can often outperform the predictive ability of a single classifier. Bagging, boosting and stacking are among the most well-established ensemble techniques. Bagging (Breiman, 1996) is the simplest of the three; it consists of an ensemble of models of the same type that are trained using different training sets, with the models' predictions being aggregated by majority voting; this technique is especially useful when the training algorithm of the models is unstable. Boosting (Schapire, 1990) is more complex; it iteratively creates new models that are more accurate at predicting the instances that were misclassified by previously trained models, by reweighting the data after each model is trained: instances that were misclassified gain weight, while instances that were classified correctly lose weight; by changing these weights, the next model to be trained will give more importance to the most problematic instances. The resulting ensemble performs classification by weighted voting, with the weight of each vote being proportional to the accuracy of each model. This method is most commonly used with weak learners, i.e., simple and relatively inaccurate models like decision stumps; boosting can frequently turn these weak learners into a single strong learner (the ensemble). Stacking (Wolpert, 1992) differs from both bagging and boosting in that it combines models of different types; the votes of the models in the ensemble are aggregated by a meta-learner, i.e., a data mining model that learns how best to combine the predictions of these models. Besides bagging, boosting and stacking, many other ensemble techniques have been proposed in data mining literature. Since none of them was custom-made for financial prediction, we decided to come up with our own ensemble creation strategy. We did it step-by-step, starting with a simple ensemble composed of the seven models that were previously tested. This ensemble was integrated into an architecture component named "prediction module", which is responsible for feeding the instances to the models, and for aggregating their predictions into a single forecast. A simple USD/JPY trading agent was built around this module, in compliance with the architecture presented in Figure 9. As depicted in this figure, the agent receives information updates about the financial instrument from the trading environment, and uses that information to periodically compile new test instances; then, it classifies those instances with the ensemble – which, in practical terms, is equivalent to making predictions regarding the direction of the price – and uses these predictions to open new trades. Our first implementation of the prediction module aggregated the models' predictions by simple majority voting. Let p_M^{t+1} represent the set of class predictions of the M models in the ensemble for trading period $t+1$, or $p_M^{t+1} = (p_1^{t+1}, \dots, p_m^{t+1})$; the ensemble prediction for period $t+1$ is given by:

$$EnsPrediction(p_M^{t+1}) = \begin{cases} UP \;\; if \;\; |U| > |D| \\ DOWN \;\; if \;\; |U| < |D| \end{cases} \tag{1}$$

$$where\ U = \left\{ i \in \mathbb{N}^*: p_i^{t+1} = UP \right\} \ and \ D = \{ i \in \mathbb{N}^*: p_i^{t+1} = DOWN \}.$$

This aggregate function implies that the predictions of the models have the same weight, therefore the ensemble class prediction for each trading period is simply the class with the biggest number of votes. Using this function, we made the USD/JPY agent simulate trades for the 2,510 out-of-sample instances. Its cumulative return throughout the simulation period is shown in Figure 10, compared with the returns of the best and the worst trading bots that were tested previously. At the end of this period, the agent had a RMD ratio of 2.23, and a 0.0131% return per trade. If we compare these results with those of the simpler trading bots, we can conclude that the agent's performance was average. This is more or less what we expected – the ensemble did not perform as well as its best model, nor as bad as its worst, because it averaged the individual performances of the models (to a certain extent). We believe this is an improvement: in the long run, the ensemble-

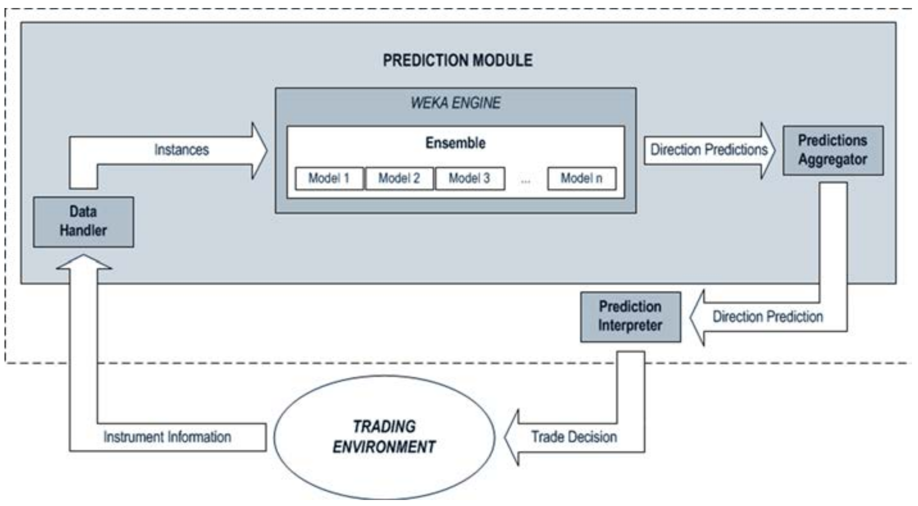


Figure 9. Trading agent architecture based on the prediction module.

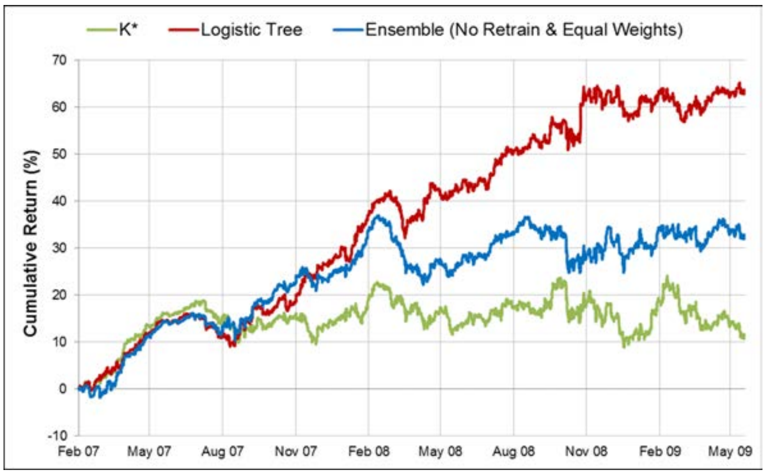


Figure 10. Gross cumulative return of the USD/JPY trading agent using an ensemble of models with equal weights.

based prediction mechanism should be more robust than the single model mechanisms, because even if some of its models become out-of-sync with the market, their poor accuracy may still be mitigated by the contributions of the other models. The simpler bots do not possess this redundancy, because they base their trading decisions on the forecasts of a single data mining model, which can become unreliable at any time.

The ensemble used in the previous trading simulation is composed of models of entirely different types, among which decision trees, lazy classifiers and regression models. Being this different, it is possible that some will be more accurate predictors when the instrument's price is trending upward, while others might perform better when it is trending downward. Consider the data in Table 2. This table breaks down the accuracy and return per trade of the seven trading bots that were tested before, according to the type of trade executed (long trades correspond to *UP* predictions, while short trades correspond to *DOWN* predictions). The RIPPER rule learner, for example, was able to predict price drops with 60.8% accuracy, but could only predict price increases with 51.6% accuracy. The K* regression model, on the other hand, was less accurate predicting the drops (52.8%), and more accurate predicting price increases (53.0%). Suppose the RIPPER classification model predicts a price decrease for a given trading period, while the K* regression

Table 2. Accuracy and return per trade of the USD/JPY trading bots (excluding trading costs).

Model	*Acc (%)*	*Long Acc (%)*	*Short Acc (%)*	*Ret/Trade (%)*	*Long Ret/Trade (%)*	*Short Ret/Trade (%)*
K* (classification)	52.4	52.0	53.0	0.0046	-0.0016	0.0152
C4.5 Decision Tree	53.2	52.0	57.1	0.0140	0.0048	0.0440
RIPPER Rule Learner	52.9	51.6	60.8	0.0179	0.0066	0.0844
Naïve Bayes	52.1	51.8	52.5	0.0118	0.0044	0.0226
Logistic Model Tree	54.3	53.7	55.2	0.0254	0.0160	0.0388
K* (regression)	52.9	53.0	52.8	0.0170	0.0101	0.0245
Supp. Vector Machine	53.4	53.3	53.5	0.0223	0.0146	0.0313

model predicts a price increase; in this scenario, it would make sense to accept the RIPPER's forecast, because in the past this model was more capable of predicting price declines than the K* was of predicting price gains; however, if the opposite happened, it would be better to go with the forecast of the K*. Building a prediction mechanism that capitalizes on these accuracy differences might improve the profitability of the agents. In order to test this assumption, we modified the implementation of the prediction module: it still used an ensemble with the exact same models, but each model's vote now had its own specific weight, and these weights were updated periodically over time. We started by making the weights proportional to the models' accuracy. Let p_N^m represent the sequence of class predictions from trading period $t - N + 1$ till period t for model m, or $p_N^m = (p_{t-N+1}^m, \dots, p_t^m)$, and $r_N^m = (r_{t-N+1}^m, \dots, r_t^m)$ represent the returns that would have been obtained if those predictions were utilized to open trades. The long and short accuracy factors, to be used as model m's vote weights in period $t + 1$, are:

$$LongAF(p_N^m, r_N^m) = \frac{|U'|}{|U|} - 0.5, \qquad (2)$$

$$ShortAF(p_N^m, r_N^m) = \frac{|D'|}{|D|} - 0.5, \qquad (3)$$

$$\textit{where } U = \{i \in \mathbb{N}^*\colon p_i^m = UP\},\ U' = \{i \in U\colon r_i^m > 0\},$$
$$\textit{and } D = \{i \in \mathbb{N}^*\colon p_i^m = DOWN\},\ D' = \{i \in D\colon r_i^m > 0\}.$$

The prediction module calculates these values for each model whenever a trading period ends, and another is about to begin. Once all the models have made their predictions for the new period, the module computes the ensemble prediction by weighted voting. Notice this strategy is somewhat similar to Barbosa and Torgo's (2006), only they used just one type of weight, which was based on the *F*-measure (Rijsbergen, 1979). Let p_M^{t+1} represent the set of class predictions of the M models in the ensemble for trading period $t + 1$, or $p_M^{t+1} = (p_1^{t+1}, \dots, p_m^{t+1})$, $l_M^t = (l_1^t, \dots, l_m^t)$ represent the long accuracy factors of the M models (calculated at the end of period t with Equation 2 using the last N trades), and $s_M^t = (s_1^t, \dots, s_m^t)$ represent the short accuracy factors (calculated the same way, but with Equation 3). The ensemble forecast of the prediction module for period $t + 1$ will be:

$$EnsPrediction(p_M^{t+1}, l_M^t, s_M^t) = \begin{cases} UP \ \ if \ \sum_{i \in U} l_i^t > \sum_{i \in D} s_i^t \\ DOWN \ \ if \ \sum_{i \in U} l_i^t < \sum_{i \in D} s_i^t \\ NONE \ \ if \ \sum_{i \in U} l_i^t = \sum_{i \in D} s_i^t \end{cases} \quad (4)$$

$$\text{where } \mathrm{U} = \left\{\mathrm{i} \in \mathbb{N}^*\colon \mathrm{p}_\mathrm{i}^{\mathrm{t}+1} = \mathrm{UP} \wedge \mathrm{l}_\mathrm{i}^\mathrm{t} > 0\right\} \text{ and } \mathrm{D} = \{\mathrm{i} \in \mathbb{N}^*\colon \mathrm{p}_\mathrm{i}^{\mathrm{t}+1} = \mathrm{DOWN} \wedge \mathrm{s}_\mathrm{i}^\mathrm{t} > 0\}.$$

According to this aggregate function, the weight of a model's vote will either be its long accuracy factor, if it predicts a price increase, or its short accuracy factor, if it predicts a price decrease. If the sum of vote weights of the models predicting the price will go up is greater than the sum of vote weights of the models predicting it will go down, then the ensemble prediction is class *UP*; if the reverse happens, the ensemble predicts a price decrease; if the sums are exactly the same, the ensemble does not return a prediction. Notice that only the models with a positive vote weight are considered in the calculation of the ensemble forecast. This means that, if all the models were very inaccurate when forecasting the last N trades, the prediction module will not output a prediction for the next trading period (because all the weights will be negative).

This method of calculating the weights and aggregating the votes is best explained with an example. Imagine that a data mining model made six *UP* and four *DOWN* predictions in the last ten trading periods; four of the *UP* forecasts were accurate, i.e., the model's classification for the instances was class *UP*, and the price increased in the corresponding periods (therefore, if these individual forecasts were used to open trades, four *UP* predictions would have originated positive returns, while two would have ended with losses). Now imagine that, of the four *DOWN* predictions, only one was accurate. In this scenario, the model's long accuracy factor at that instant would be 0.17 ($\frac{4}{6} - 0.5$), while its short accuracy factor would be -0.25 ($\frac{1}{4} - 0.5$); we subtract 0.5 from the proportion of accurate predictions because the minimum accuracy we accept is 50% (which even a simple coin-flipping mechanism should be able to achieve). If this model makes an *UP* prediction for the next trading period, the weight of its vote will be 0.17; if, on the other hand, it predicts class *DOWN*, its vote weight will be -0.25 – since this weight is negative, it will be filtered out by the aggregate function and excluded from the calculation, which in practical terms means that the model will be ignored. The rationale behind this strategy to compute the ensemble prediction is that it allows the agent to ignore the forecasts of models that were inaccurate in the recent past; also, it makes the agent "smart" enough to stop trading when all the models become inaccurate, until their precision improves.

We configured the USD/JPY trading agent to use this new implementation of the prediction module, set N to 50 (so that the weights were based on the models' performance under the most recent market conditions), and repeated the trading simulation. Its accumulated return throughout the out-of-sample period is shown in Figure 11, in comparison with the return of the simpler agent that used an ensemble with equal weights. The new agent obtained a RMD ratio of 2.31, slightly better than the ratio of the simpler version; its 0.0122% return per trade, on the other hand, was not as good. The biggest difference between the two was in the number of trades simulated. Even though the out-of-sample period consists of 2,510 instances, the new agent only performed 2,327 trades. This implies that it temporarily stopped trading during the simulation, because all of its data mining models were exhibiting poor accuracy in the recent past; that is to say, all the models' votes had negative weights, so the prediction module stopped outputting predictions. All things considered, we must conclude that, for this particular agent, using dynamic accuracy-based weights in the prediction module was not a significant improvement over using equal weights.

As we mentioned before, from a trader's perspective, profit is much more important than accuracy. Therefore, it is probably best to base the models' vote weights on their past profitability, rather than their accuracy. We can see in Table 2 that the average return per trade obtained with each standalone data mining model varied substantially according to the type of predictions made: some were more profitable with *UP* classifications (i.e., long trades), while others did better with *DOWN* classifications (short trades). The most profitable at predicting price increases was the logistic model tree, while the RIPPER rule learner was the best at shorting the currency pair. We can also verify that *UP* classifications by the K* classifier were correct 52.0% of the times, but the corresponding trades were unprofitable on average; hence, the trading agent should disregard this model whenever it does that classification. In order to allow the agent to focus on this type of

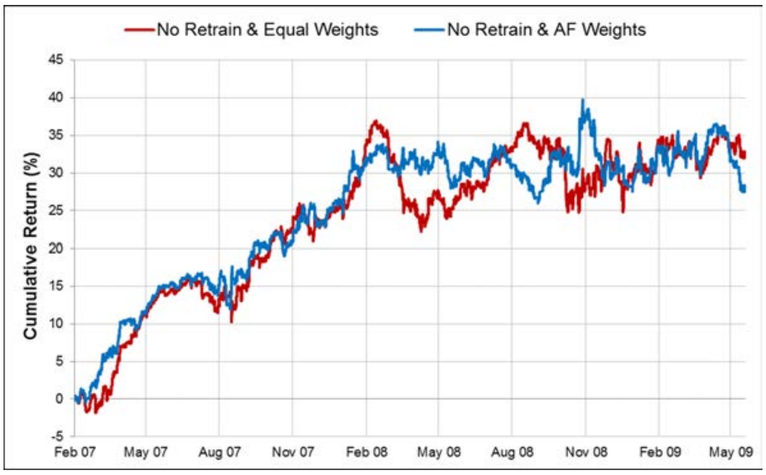


Figure 11. Gross cumulative return of the USD/JPY trading agent using an ensemble of models with dynamic accuracy-based weights.

information, rather than the models' accuracy, we introduced a new change to the prediction module. It acted exactly the same way as in the previous implementation, but the models' vote weights were now based on their individual profitability. Consider once again that p_N^m represents the sequence of class predictions from trading period $t - N + 1$ till period t for model m, and r_N^m represents the returns that would have been obtained if we opened trades according to those predictions. The long and short profit factors, to be used as model m's vote weights in period $t + 1$, are given by:

$$LongPF(p_N^m, r_N^m) = \frac{\sum_{i \in U'} r_i^m}{\sum_{i \in U''} |r_i^m|} - 1, \tag{5}$$

$$ShortPF(p_N^m, r_N^m) = \frac{\sum_{i \in D'} r_i^m}{\sum_{i \in D''} |r_i^m|} - 1, \tag{6}$$

$$where\ U' = \{i \in \mathbb{N}^*: p_i^m = UP \wedge r_i^m > 0\},\ U'' = \{i \in \mathbb{N}^*: p_i^m = UP \wedge r_i^m < 0\},$$
$$and\ D' = \{i \in \mathbb{N}^*: p_i^m = DOWN \wedge r_i^m > 0\},\ D'' = \{i \in \mathbb{N}^*: p_i^m = DOWN \wedge r_i^m < 0\}.$$

These values are calculated by the prediction module before each trade. When a model predicts a price increase, the weight of its vote is its long profit factor; when it predicts a price decrease, the weight of its vote is its short profit factor. The long profit factor is basically the ratio between the return the model would have obtained with its accurate *UP* predictions, and the return it would have lost with its inaccurate *UP* predictions, in the last *N* periods. The short profit factor is similar, only for the *DOWN* classifications in the recent past.

Let us go back to our previous scenario, where a fictitious model made four *UP* and one *DOWN* accurate predictions, versus two *UP* and three *DOWN* inaccurate classifications, in the last ten periods. Imagine that, if these forecasts were utilized to open trades, the returns obtained would be: 3.5% with the four accurate long trades, -4.5% with the two failed long trades, 4.75% with the accurate short trade, and -2.75% with the three bad short trades. This scenario implies that, even though the model was more precise at forecasting price increases than price decreases, the overall return of the long trades was negative, while the total return of the short trades was positive. The model's long profit factor, at this particular point in time, would be -0.22 ($\frac{3.5}{|-4.5|} - 1$), while the short profit factor would be 0.73 ($\frac{4.75}{|-2.75|} - 1$). Notice these weights reflect how much positive return the model would yield, for each unit of negative return produced; we use the ratio to decrease the range of the weights, and subtract 1 to create a greater differentiation between them (otherwise they would likely be very similar). If the model in our example predicts class *UP* for period $t+1$, its vote will be assigned a negative weight, and will therefore be ignored by the prediction module; a *DOWN* prediction, on the other hand, will be assigned a positive weight of 0.73. This strategy makes sense in that we are putting the emphasis on the models' ability to generate profit, rather than their ability to predict the direction of the price – even if the models are very accurate, the prediction module will not output any predictions (thus preventing the agent from trading) if that accuracy did not translated into actual profit in the last N trades.

We devised a new prediction module to test this profit-based weighting strategy. The function that aggregates the models' predictions remains the same (Equation 4), only l_M^t and s_M^t now represent the long and short profit factors of the M models in period t, instead of their accuracy factors. With the N parameter set to 50, the new version of the USD/JPY trading agent achieved in the simulation period the cumulative return shown in Figure 12. For comparison purposes, the returns obtained with the other module implementations are also displayed in this figure. The chart reveals that, for this specific financial instrument and time frame, profit-based vote weights were a better option than accuracy-based weights. The new agent achieved a RMD ratio of 3.50 and a return per trade of 0.0227%, both metrics demonstrating a significant improvement over the previous implementations of the prediction module. The agent only simulated 2,339 trades (out of 2,510 possible), which means that it temporarily stopped trading during the simulation, because all the models were making unprofitable predictions in the recent past.

So far, we have seen how a trading agent's performance varies when its prediction mechanism is altered from a single data mining model to an ensemble of models with equal or dynamic vote

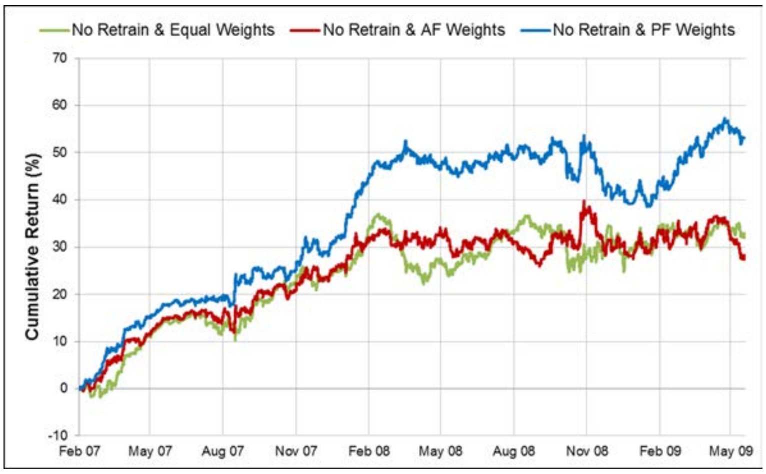


Figure 12. Gross cumulative return of the USD/JPY trading agent using an ensemble of models with dynamic profit-based weights.

weights (based on accuracy or profit). These models remained static throughout the course of the simulations, unable to learn new patterns while the trades were being performed. As we previously explained, it is extremely unlikely that the financial data used for training the models will ever contain all the information they need to keep making accurate predictions indefinitely. After each forecast, and as soon as the corresponding trade is simulated, there is always a new instance available that could be utilized to train the models. Periodically retraining them with these new instances might improve their accuracy over time; however, since some learning algorithms may be unstable, it is important to check if the retrained models perform as well as they did before retraining. We implemented a new version of the prediction module to test this retraining strategy. Before each prediction, this module splits all the available instances into two datasets: the test set, with the most recent *N* instances, and the training set, with all the rest. Using these two sets of data, the following sequence of steps is applied to each model in the ensemble:

- The model is retrained using the training set.
- The retrained model is utilized to make class predictions for all the test instances, and trades are simulated accordingly; using the results of this simulation, the overall accuracy factor of the retrained model is calculated with the following equation:

$$OverallAF(r_N^m) = \frac{|T'|}{N} - 0.5, \tag{7}$$
$$where\; T' = \{i \in \mathbb{N}^*: r_i^m > 0\}.$$

Notice that r_N^m has the same meaning as in previous equations, i.e., it is the sequence of simulated returns corresponding to model m's last N predictions.

- If the overall accuracy factor of the retrained model is greater than or equal to the overall accuracy factor of the model before retraining, then the retrained model replaces it in the ensemble. Otherwise, the retrained model is discarded, and the original is kept.

This algorithm ensures that the agent will keep learning as time goes by, because its data mining models will be periodically retrained with a bigger training set. Figure 13 illustrates this concept of a growing training set, coupled with a test set that moves like a sliding window containing only the most recent data. By using the accuracy in the last N trades (the test set) to decide if a model should be replaced with a retrained version of itself, the prediction module guarantees that, at any given point in time, the ensemble contains models that are performing as best as they can in the most recent market conditions. This method might also confer some trend-following capabilities to the agent, because the models are chosen according to test data that reflects the most recent price trend. This strategy is similar to that of Harries and Horn (1995), only they always replaced the original model with the retrained model, and the training data was part of the sliding window.

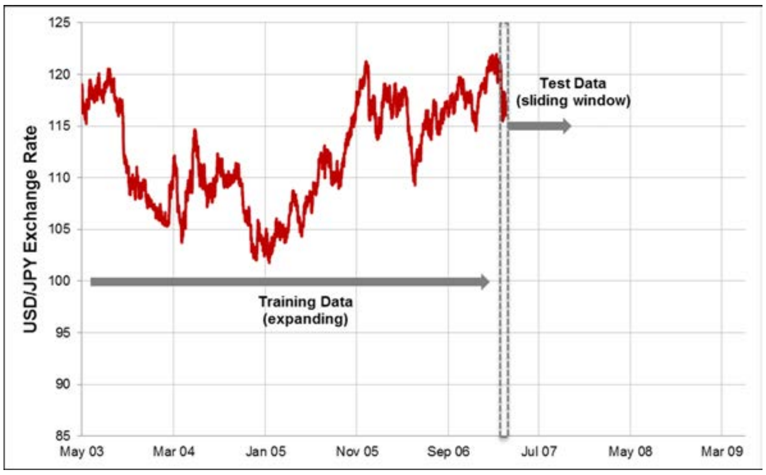


Figure 13. Split between the training and the test data at a specific point in the simulation.

We will explain our retraining mechanism with a simple example. Consider that a trading period has just ended, and the corresponding new instance has been added to the set of available data. Parameter N is set to 100, hence the data is split into a test set with the last 100 instances, and a training set with all the rest. Notice the new instance becomes a test instance, while the oldest test instance becomes a training instance. Using model m's predictions for the test data, Equation 7 is utilized to calculate its overall accuracy factor; suppose the calculated value is 0.05, meaning the model's accuracy for the last 100 periods was 55%. Next, a new model m' is trained with the training set, using the exact same algorithm, parameters and attributes that were originally used to create model m. This new model makes predictions for the same test instances, and its overall accuracy factor is calculated the same way with Equation 7. Now, if its accuracy factor is greater than or equal to that of model m (i.e., if it is at least 0.05), then m' becomes a part of the ensemble, and m is remove from it; otherwise, m' is discarded. Once this strategy has been applied to all the models in the ensemble, the prediction module is ready to output its forecast for the next trading period. This procedure entails that the models in the ensemble will be periodically replaced with more accurate versions of themselves, except when retrained models are put in the ensemble because they "overfit" the test data (which is counterproductive). To minimize the chances of this happening, the decision to replace the models could be based on cross-validation, rather than a single test set. However, this would be too computer-intensive (and consequently, the prediction mechanism would be too slow to be useful in practice); also, it would not give the agent the trend-following ability we are hoping to achieve – by selecting the models using just one test set with the most recent data, the ensemble will contain the models that performed best in the most recent past, which could prove useful in trending markets.

We implemented an agent using this new version of the prediction module. To be consistent with previous tests, N was set to 50. The agent's performance throughout the simulation period is shown in Figure 14, compared with the return of the agent with the simpler prediction module (which did not retrain the models). We specified equal vote weights in this simulation, i.e., the ensemble prediction was decided by simple majority voting (Equation 1). Much to our surprise, the results we got with the new agent were considerably worse than those of the simpler agent: it achieved a RMD ratio of 1.48, and a return per trade of just 0.0112%. Not impressive, by any standard.

Once again, we attempted to improve the agent's performance by making its prediction module focus on profit, rather than accuracy. Instead of using the overall accuracy factor to decide when a retrained model should become a part of the ensemble, we made it use the overall profit factor, for which we defined the following equation:

$$OverallPF(r_N^m) = \frac{\sum_{i \in T'} r_i^m}{\sum_{i \in T''} \left| r_i^m \right|} - 1, \qquad (8)$$

$$where\ T' = \{i \in \mathbb{N}^*:\ r_i^m > 0\}\ and\ T'' = \{i \in \mathbb{N}^*:\ r_i^m < 0\}.$$

This new version of the prediction module ensures that a retrained model m' will only replace the original model m in the ensemble if it is determined that m' would have been at least as profitable as m in the most recent N trades. The agent's cumulative return throughout the simulation period,

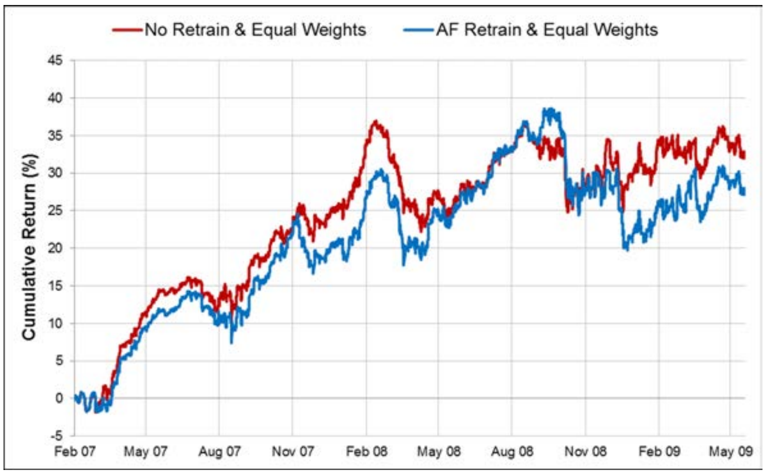


Figure 14. Gross cumulative return of the USD/JPY trading agent using an ensemble with periodical retraining and replacement of models based on accuracy.

using this new strategy, is displayed in Figure 15. Also shown are the returns obtained with the other two versions of the prediction module (no retraining, and retraining with replacement based on accuracy). We can see in the chart that the new agent was more profitable than the alternatives. It achieved a RMD ratio of 1.61 and a return per trade of 0.0142%, which is slightly better than the performance of the accuracy-based solution. Still, this improvement is not big enough to justify the computational overhead of periodically retraining the models.

Up to this point, we have experimented with two different strategies for improving the performance of a trading agent. First, we tried to create an agent that could adapt to different markets conditions, by assigning dynamic vote weights to the data mining models in its prediction module. Next, we tried to create an agent that would keep learning over time, by retraining the models in its ensemble before each prediction. Clearly, the next logical step is to create an agent with both capabilities. To accomplish this, a new prediction module was implemented. We made it use dynamic vote weights based on the long and short accuracy factors (Equations 2 and 3), and retrain and replace models based on the overall accuracy factor (Equation 7); the aggregation of the models' votes was done with Equation 4. With this new module, the agent obtained the cumulative

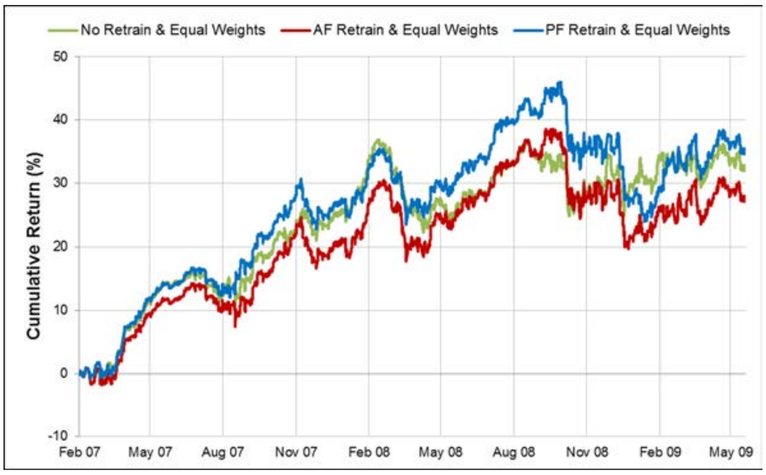


Figure 15. Gross cumulative return of the USD/JPY trading agent using an ensemble with periodical retraining and replacement of models based on profit.

return displayed in Figure 16; plotted in comparison is the return of the simpler agent, which used equal vote weights and did not retrain the models. Overall, the new strategy yielded promising results: the agent had a RMD ratio of 4.70 and a return per trade of 0.0183%, both measures demonstrating a significant improvement over the less complex agent. We expected to get even better results by focusing on the profit, instead of the accuracy. To accomplish this, we created yet another prediction module. This time around, it utilized dynamic vote weights based on the models' long and short profit factors (Equations 5 and 6), and retrained and replaced models according to their overall profit factor (Equation 8); vote aggregation was also done with Equation 4. The return of the new agent is presented in Figure 17, in comparison with the returns obtained with previous implementations of the prediction module; we can clearly verify that it performed much better than any of the other solutions.

Table 3 summarizes the simulation results for all the different prediction module implementations that were tested. Without a doubt, this last profit-based strategy was the one that showed the greatest potential. Its RMD ratio was 5.35, and its return per trade was 0.0260%, a performance that soundly beat all the other solutions. This led us to believe that this forecasting

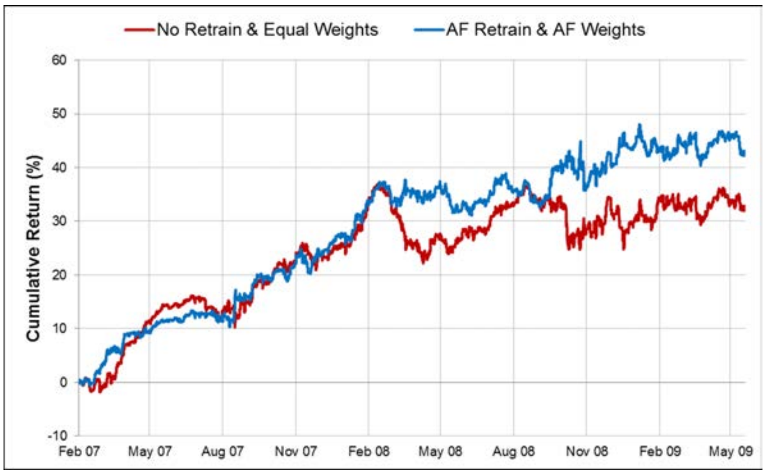


Figure 16. Gross cumulative return of the USD/JPY trading agent using an ensemble with retraining and dynamic vote weights based on accuracy.

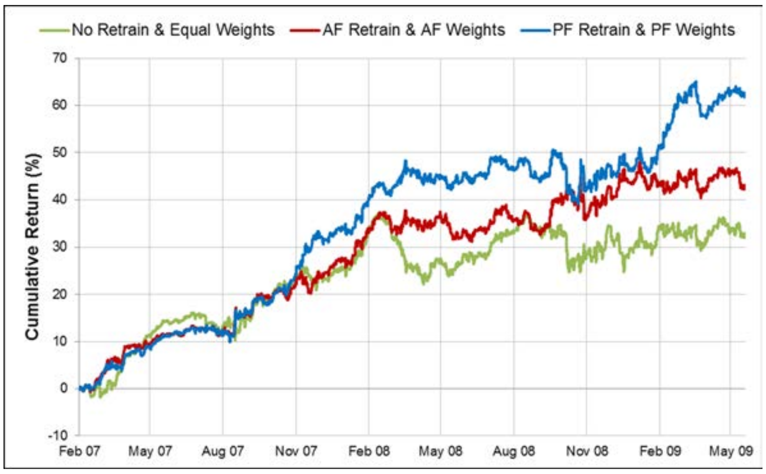


Figure 17. Gross cumulative return of the USD/JPY trading agent using an ensemble with retraining and dynamic vote weights based on profit.

Table 3. Simulation results of the USD/JPY trading agent using different prediction module implementations (excluding trading costs).

Prediction Module	*Ret (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Acc (%)*	*Trades*
Equal weights	32.9	14.8	2.23	0.0131	53.4	2,510
Dynamic weights (AF)	28.4	12.3	2.31	0.0122	52.6	2,327
Dynamic weights (PF)	53.1	15.2	3.50	0.0227	53.0	2,339
Retrain (AF)	28.0	18.9	1.48	0.0112	53.1	2,510
Retrain (PF)	35.5	22.1	1.61	0.0142	53.6	2,510
Retrain & Dynamic weights (AF)	43.1	9.2	4.70	0.0183	52.5	2,362
Retrain & Dynamic weights (PF)	61.8	11.5	5.35	0.0260	52.2	2,375

mechanism might be good enough to be used as a building block in the architecture of intelligent trading agents. Still, more empirical evidence was needed to support this decision.

In order to determine if similar success could be achieved with a different financial instrument, we repeated all the experiments with an agent that was configured to day trade the ADBE stock. Eleven data mining models were trained, using historical data for the period between August of 1986 and October of 2005, which corresponds to about 4,850 instances; the 50 instances that comprise the period from November of 2005 till the second week of January of 2006 were used for testing; the subsequent data, up to the second week of May of 2009, was reserved for the trading simulations (a total of 828 out-of-sample instances). The price of the stock throughout these periods is presented in Figure 18.

Following the same strategy as before, each of the eleven data mining models was utilized to implement a simple trading bot, based on the architecture shown in Figure 4. The models were trained to make daily predictions regarding the direction of the price of the ADBE stock; more specifically, they tried to predict if the difference between the stock's closing and opening prices in the following day was going to be positive or negative. For each out-of-sample instance, if the model predicted a positive change, i.e., a price increase from the market's opening to the close, the bot simulated buying at the open and selling at the close. If the prediction was for a price decrease throughout the day, the bot simulated going short at the open and covering at the close. Just like in previous tests, we specified a fixed trade size in these simulations. The cumulative returns of the eleven bots in the out-of-sample period are displayed in Figures 19 through 21, and summarized in Table 4. Our results reveal that only three bots (with the models Naïve Bayes, K* and least median squared regression) were profitable at the end of the simulation period. Some of them (fuzzy lattice

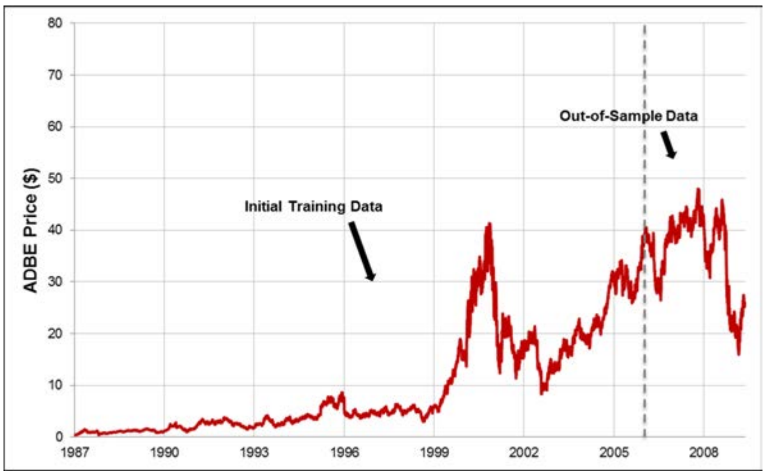


Figure 18. Historical ADBE stock prices.

reasoning, classification and regression tree, pace regression) experienced massive losses. This is troubling, especially if we consider that the returns shown do not even reflect the trading costs (which, going by the number of trades, would make the losses much worse). One thing stands out the most when we look at the charts: all the returns experienced a significant increase in volatility after September of 2008. This was the month when investment bank Lehman Brothers filed for bankruptcy, because of the subprime mortgage crisis; many other financial institutions followed suit shortly thereafter. This event completely changed the dynamics of the stock market, with global indices suffering crashes of historic proportions. The ADBE stock price was dragged down with the rest of the market, as we can see in Figure 18. Consequently, several trading bots experienced massive losses. This corroborates our belief that a prediction mechanism that is based on a standalone data mining model cannot be used to trade autonomously, because sooner or later it will be affected by one of these catastrophic events. Even if the model is able to output profitable predictions for several years in a row, it may end up losing all its investment capital when a black swan event occurs – this is exactly what happened to the trading bot with the pace regression model: it did ok for well over two years, but was overwhelmed by the volatility when the subprime crisis hit the markets, and ended up losing almost half of its capital. The bot with the K*

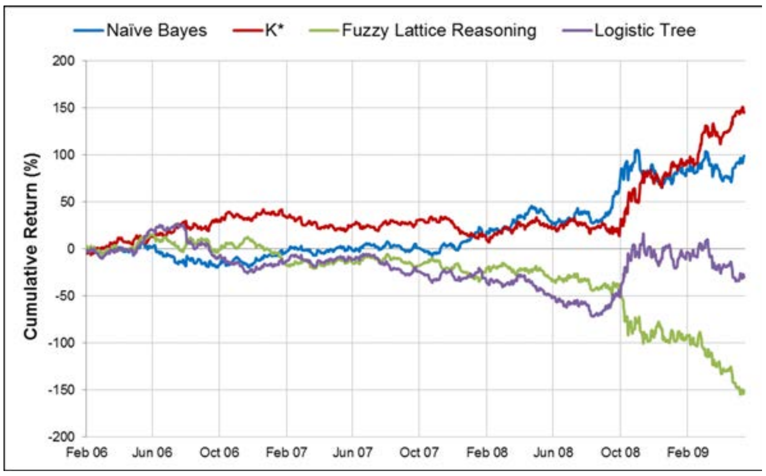


Figure 19. Gross cumulative return of the ADBE trading bots with standalone classification models.

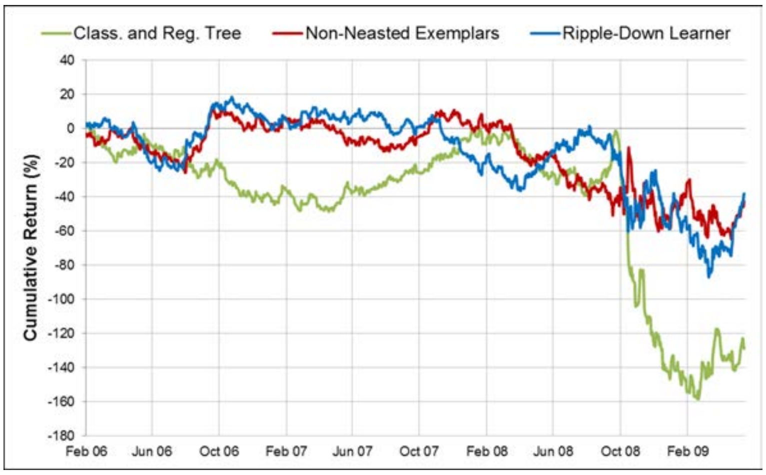


Figure 20. Gross cumulative return of the ADBE trading bots with other standalone classification models.

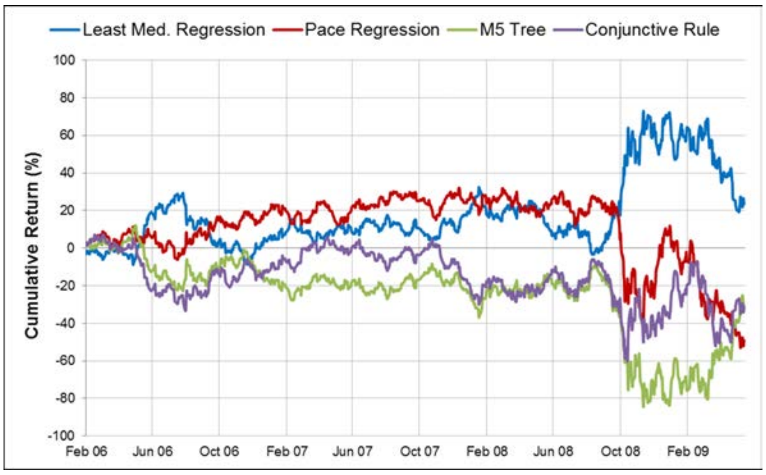


Figure 21. Gross cumulative return of the ADBE trading bots with standalone regression models.

classification model, on the other hand, did quite well in the same situation. But that does not mean it would be a good idea to let it trade real funds: first of all, it is impossible to know if the K* model will continue to be profitable going forward; also, the bot's track record is not flawless – looking at the cumulative return curve in Figure 19, we verify that its return peaked in the beginning of 2007, after which it took it almost two years to overcome that high-water mark. Losing money for two years in a row is a very big drawback; if this happened in real life, we would certainly be questioning the bot's competence after a while, and might end up getting rid of it before it had the chance to recover the losses. Most other bots exhibit the same problem (extended

Table 4. Simulation results of the ADBE trading bots (excluding trading costs).

Model	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Trades*
Naïve Bayes	98.9	40.2	2.46	0.1195	50.8	828
Least Median Squared Linear Regression	23.8	53.5	0.44	0.0287	48.1	828
Pace Regression	-49.4	85.0	-0.58	-0.0597	48.4	828
K*	145.2	34.9	4.17	0.1754	51.8	828
Fuzzy Lattice Reasoning	-151.2	170.2	-0.89	-0.1826	46.3	828
Logistic Model Tree	-29.9	98.6	-0.30	-0.0361	45.8	828
M5 Model Tree	-32.8	96.3	-0.34	-0.0396	50.2	828
Classification and Regression Tree	-128.6	161.5	-0.80	-0.1553	48.1	828
Conjunctive Rule	-31.4	67.4	-0.47	-0.0379	49.3	828
Non-Nested Generalised Exemplars	-42.7	76.7	-0.56	-0.0516	50.2	828
Ripple-Down Rule Learner	-38.4	105.5	-0.36	-0.0463	51.1	828

periods of time where the cumulative return moves downward or sideways), which would be nerve-racking to experience in real-time, in addition to representing a substantial opportunity cost. All things considered, we must conclude that none of the bots demonstrated much trading talent.

The fact that the out-of-sample instances were extracted from a period with extreme volatility spikes makes them the perfect dataset for testing a trading agent's ability to adapt to changes in market dynamics. In order to do so, we used the architecture shown in Figure 9 to create an agent to trade the ADBE stock. At the centre of its prediction module was an ensemble composed of the eleven data mining models that were utilized by the trading bots. Just like before, we experimented with various prediction module implementations. The results obtained with the different solutions in the out-of-sample period are displayed in Figures 22 to 25, and summarized in Table 5.

Figure 22 reveals that, while the performance of the simplest ensemble with the eleven models was marginally better than that of its worst model, it was still embarrassingly bad: the agent lost

93.2% of the capital, and that is without even considering the trading costs. We were expecting this, to some extent, because the vast majority of the models in the ensemble were not very good on their own.

In Figure 23 we can see that using accuracy or profit-based vote weights in the ensemble considerably improved the agent's return, compared to aggregating the votes by simple majority voting; in these experiments, the profit-based solution yielded the best performance. This represents more empirical evidence in favour of our method of aggregating votes using dynamic

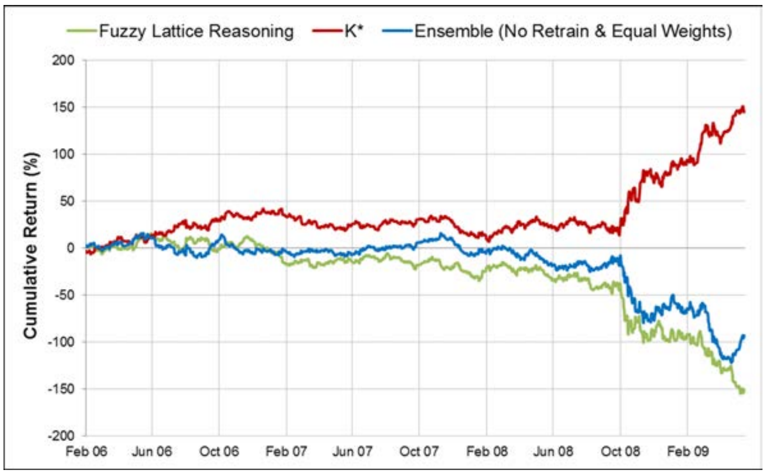


Figure 22. Gross cumulative return of the ADBE trading agent using an ensemble of models with equal weights.

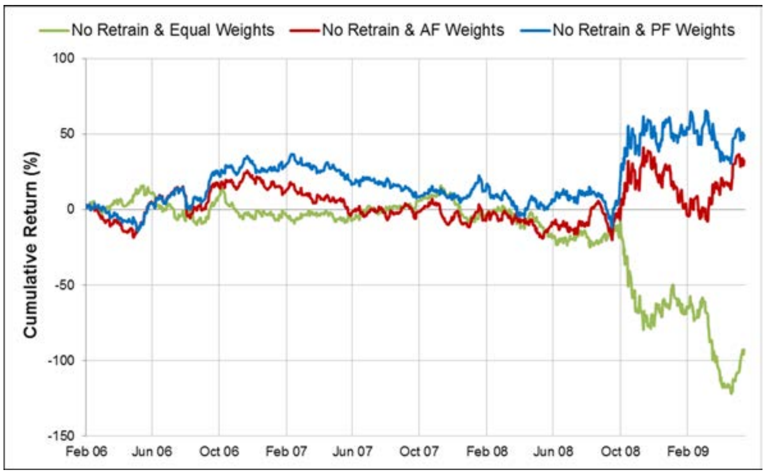


Figure 23. Gross cumulative return of the ADBE trading agent using an ensemble of models with accuracy or profit-based dynamic vote weights.

weights. Similarly, Figure 24 shows that both the accuracy and the profit-based model replacement techniques improved the performance of the agent, compared to not doing the retraining; this time around, the accuracy-based solution performed best. Overall, the results of our tests support our conviction that dynamic vote weights and model retraining can increase the skill level of the trading agent, by making it more resilient to market swings.

Figure 25 indicates that the agent performs even better with a prediction module that implements both techniques simultaneously, but only if the models' replacement and vote weights are based on their short term profitability. These results are similar to the ones we got with the USD/JPY trading agent. For both agents, the final profit-based solution – that enables them to keep learning over time (by retraining the models) and to adapt to changes in market dynamics (by updating the models' vote weights) – is the one that yielded the best performance. For this reason, we decided to use this particular version of the prediction module as a building block in our trading agent architecture. This module will allow the intelligent agents to decide when to buy or short sell a financial instrument, thus taking care of one of the basic autonomy requirements that were previously established for these agents.

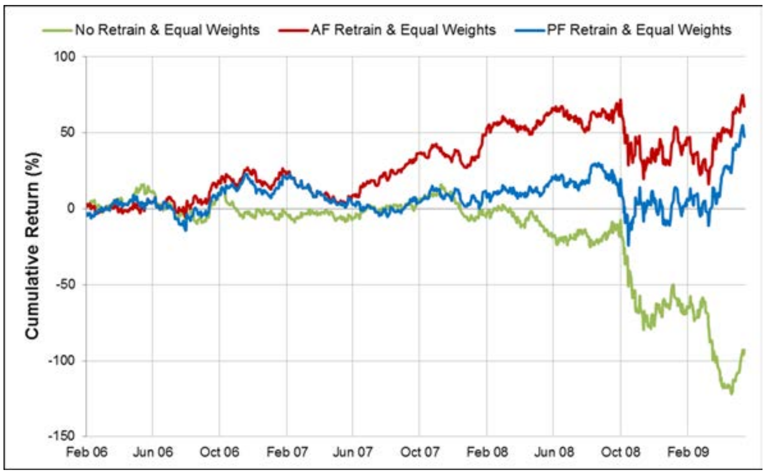


Figure 24. Gross cumulative return of the ADBE trading agent using an ensemble with accuracy or profit-based periodical retraining and replacement of models.

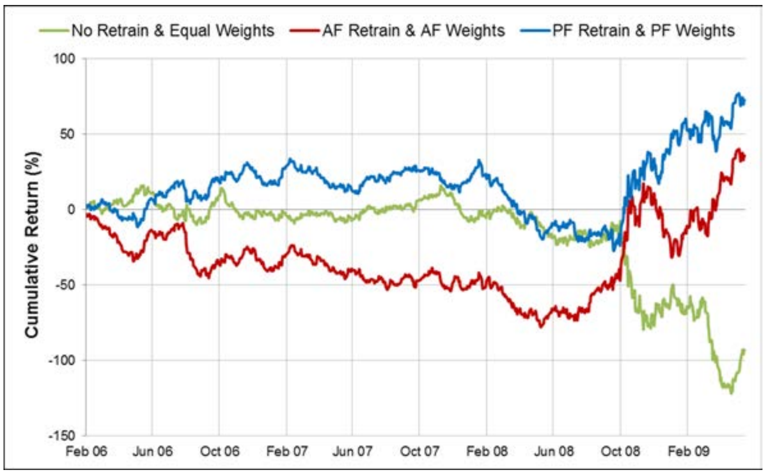


Figure 25. Gross cumulative return of the ADBE trading agent using an ensemble of models with accuracy or profit-based dynamic vote weights and retraining.

Table 5. Simulation results of the ADBE trading agent using different prediction module implementations (excluding trading costs).

Prediction Module	*Ret (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Acc (%)*	*Trades*
Equal weights	-93.2	137.8	-0.68	-0.1125	48.1	828
Dynamic weights (AF)	32.1	48.5	0.66	0.0388	50.9	827
Dynamic weights (PF)	49.2	48.5	1.02	0.0595	49.6	828
Retrain (AF)	67.6	55.4	1.22	0.0817	52.4	828
Retrain (PF)	47.7	53.7	0.89	0.0576	53.1	828
Retrain & Dynamic weights (AF)	35.6	74.9	0.48	0.0429	48.8	828
Retrain & Dynamic weights (PF)	72.4	60.9	1.19	0.0874	51.0	828

The pseudocode detailing the implementation of the USD/JPY and the ADBE trading agents (which were based on the architecture in Figure 9) is listed in Algorithm 2 and depicted in the UML sequence diagram in Figure 26. The inner workings of the prediction module that was picked for future use (i.e., the one with profit-based dynamic weights and model retraining) is described in Algorithm 3, and the corresponding diagram is shown in Figure 27. We need to emphasize that, according to this algorithm, the decision to replace a model in the ensemble with a new version of itself (trained with more data) is based on the simulated profitability achieved with the test data. This signifies that the module is selecting the models by looking at how well they perform with the test instances. This is not a good data mining strategy. Just because a model made profitable forecasts for a small set of test data, that does not mean it will be able to do so for new, unseen data. Put another way, a model that cannot generalize well from the training data might make profitable predictions for a few test instances due to chance alone. Therefore, some subpar models that "overfit" the test data might end up in the agent's ensemble. But this problem is mitigated by the way the prediction module works: first of all, if a model is indeed a bad predictor, the prediction module will progressively decrease the weight of its vote in the forecasts of the ensemble (it may even set it to zero, so that the model's predictions are completely ignored);

```
Algorithm TradingAgent_v1
Inputs:
    ticker          // instrument to trade
    amount          // amount to invest in each trade
    ensemble        // ensemble of data mining model that will do the predictions
    N               // test set size

BEGIN
    tradeOpen ← FALSE
    Repeat
        confirmation ← wait_for_end(period)             // wait for the current trading period to end
        If tradeOpen = TRUE Then                        // if the prediction module made a forecast for that period:
          confirmation ← close_trade(tradeID)           //     close open trade
          tradeOpen ← FALSE
        EndIf
        periodData ← get_financial_data(ticker,period)  // get instrument's financial data for the period
        confirmation ← add_to_data(ticker,periodData)   // add period data to the database

        // --- PREDICTION MODULE --- //
        class ← predict_next_class(ticker,ensemble,N)   // predict instrument's price direction for next period (Algorithm 3)

        If class = UP Then
            tradeID ← buy_instrument(ticker,amount)     // buy if prediction module outputs class UP
            tradeOpen ← TRUE
        ElseIf class = DOWN Then
            tradeID ← short_instrument(ticker,amount)   // short sell if prediction module outputs class DOWN
            tradeOpen ← TRUE
        EndIf
    EndRepeat
END
```

Algorithm 2. Pseudocode for a trading agent based on the prediction module.

secondly, the module will continuously try to replace an unprofitable model with a more profitable retrained version of itself, and hence it should not be stuck with bad models forever. This flaw in the module's algorithm actually demonstrates its greatest strengths:

- it can ignore the predictions of inaccurate models;
- it can replace bad predictors in the ensemble as time goes by;
- it can stop making forecasts temporarily, if all the models suddenly become unprofitable predictors.

Obviously, the module's ability to spot bad models is not instantaneous; only after several inaccurate forecasts will it be capable of determining that a given data mining model is out-of-sync with the market. Still, even if there is some lag, bad models do end up being ignored or replaced sooner or later, and this process is key to the agent's autonomy.

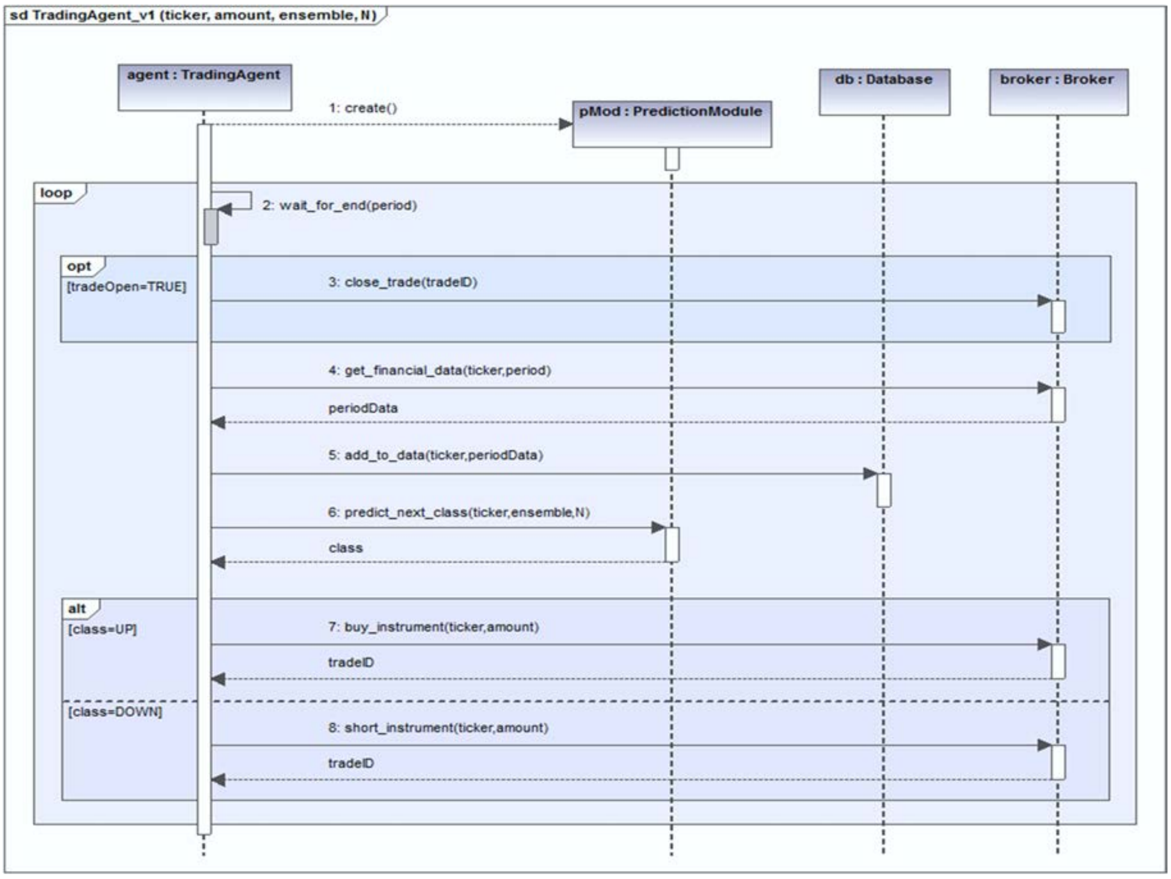

sd TradingAgent_v1 (ticker, amount, ensemble, N)
agent : TradingAgent
pMod : PredictionModule
db : Database
broker : Broker
1: create()
loop
2: wait_for_end(period)
opt
[tradeOpen=TRUE]
3: close_trade(tradeID)
4: get_financial_data(ticker,period)
periodData
5: add_to_data(ticker,periodData)
6: predict_next_class(ticker,ensemble,N)
class
alt
[class=UP]
7: buy_instrument(ticker,amount)
tradeID
[class=DOWN]
8: short_instrument(ticker,amount)
tradeID

Figure 26. UML sequence diagram for a trading agent based on the prediction module.

```
Algorithm PredictionModule_PredictNextClass
Inputs:
    ticker          // instrument to trade
    ensemble        // ensemble of data mining models that will perform the classification
    N               // test set size
Outputs:
    class           // class prediction for the next period

BEGIN
    historicalData ← get_data(ticker)                                   // get instrument's historical data from database
    totalUP,totalDOWN ← 0
    ForEach model In ensemble                                           // for each model:
        allInstances ← convert_to_instances(historicalData,model)       //   convert data into instances for the model
        instance ← get_last(allInstances)                               //   get last instance (which has no class value)
        confirmation ← remove_last(allInstances)                        //   remove last instance from the list
        testSet, trainingSet ← split(allInstances,N)                    //   split instances into test set and training set
        predictions,returns ← simulate_trades(model,testSet)            //   simulate trades for the test set
        overallPF,longPF,shortPF ← profit_factors(predictions,returns)  //   calculate profit factors (Equations 5, 6 and 8)
        model* ← retrain(model,trainingSet)                             //   retrain model using the training set
        predictions,returns ← simulate_trades(model*,testSet)           //   simulate trades using the new model
        overallPF*,longPF*,shortPF* ← profit_factors(predictions,returns) // calculate profit factors (Equations 5, 6 and 8)
        If overallPF* ≥ overallPF Then                                  //   if the new model was more profitable:
            ensemble ← remove_from_ensemble(model)                      //     remove original model from the ensemble
            ensemble ← insert_in_ensemble(model*)                       //     insert new model in the ensemble
            class ← classify_instance(instance,model*)                  //     predict price direction for next period
            If class = UP Then                                          //     if class predicted is UP:
                totalUP ← totalUP + max(longPF*,0)                      //       use the longPF* as the vote weight
            Else                                                        //     else:
                totalDOWN ← totalDOWN + max(shortPF*,0)                 //       use the shortPF* as the vote weight
            EndIf
        Else                                                            //   else if the original was more profitable:
            class ← classify_instance(instance,model)                   //     predict price direction for next period
            If class = UP Then                                          //     if class predicted is UP:
                totalUP ← totalUP + max(longPF,0)                       //       use the longPF as the vote weight
            Else                                                        //     else:
                totalDOWN ← totalDOWN + max(shortPF,0)                  //       use the shortPF as the vote weight
            EndIf
        EndIf
    EndForEach                                                          // Ensemble forecast calculation (Equation 4):
    If totalUP > totalDOWN Then                                         // if sum of UP vote weights is greater:
        class ← UP                                                      //   ensemble forecast is class UP
    ElseIf totalUP < totalDOWN Then                                     // else if sum of DOWN weights is greater:
        class ← DOWN                                                    //   ensemble forecast is class DOWN
    Else                                                                // else:
        class ← NONE                                                    //   ensemble does not make a forecast
    EndIf
    RETURN class                                                        // output the prediction
END
```

Algorithm 3. Pseudocode for the prediction module's classification task.

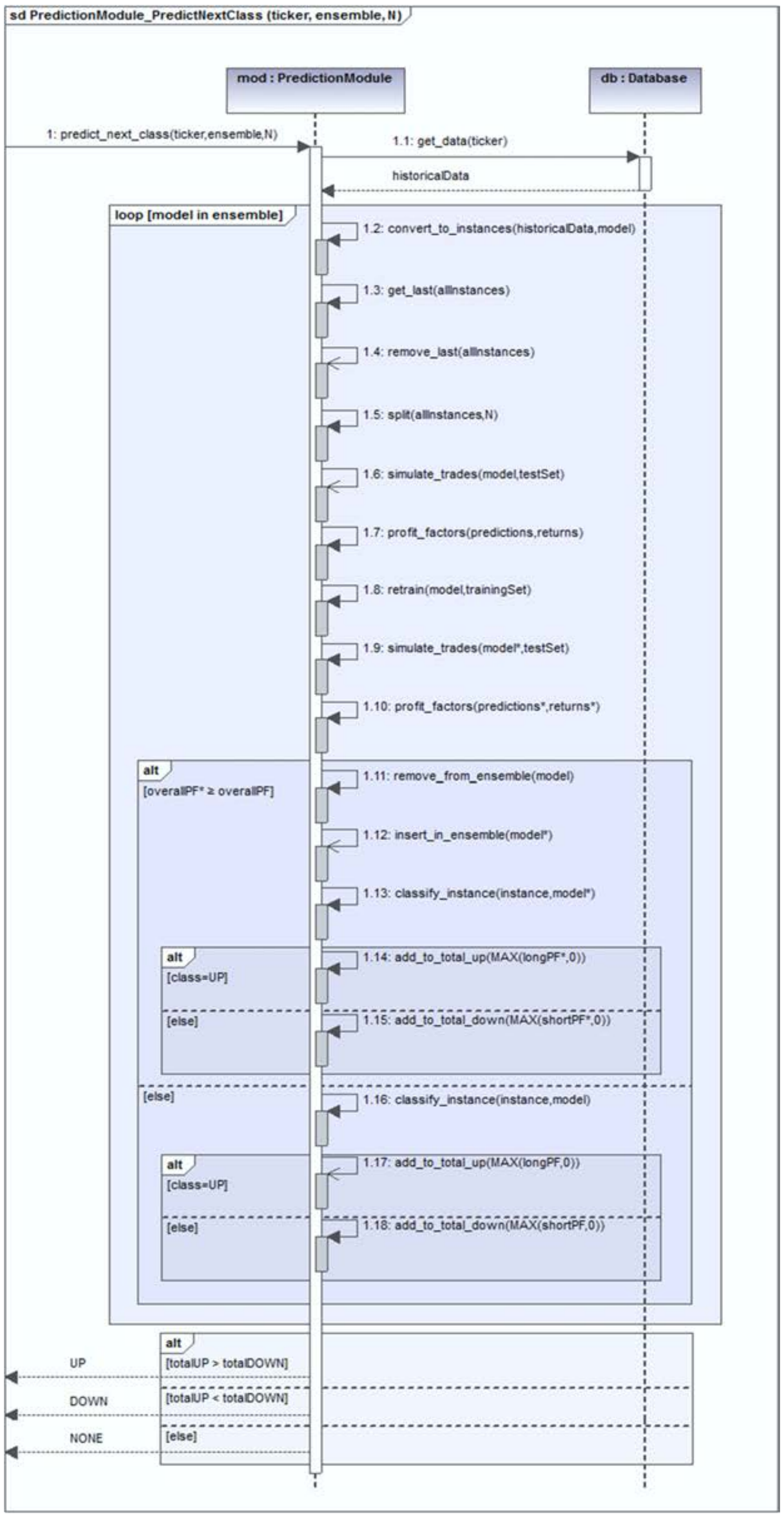

sd PredictionModule_PredictNextClass (ticker, ensemble, N)
mod : PredictionModule
db : Database
1: predict_next_class(ticker,ensemble,N)
1.1: get_data(ticker)
historicalData
loop [model in ensemble]
1.2: convert_to_instances(historicalData,model)
1.3: get_last(allInstances)
1.4: remove_last(allInstances)
1.5: split(allInstances,N)
1.6: simulate_trades(model,testSet)
1.7: profit_factors(predictions,returns)
1.8: retrain(model,trainingSet)
1.9: simulate_trades(model*,testSet)
1.10: profit_factors(predictions*,returns*)
alt
[overallPF* ≥ overallPF]
1.11: remove_from_ensemble(model)
1.12: insert_in_ensemble(model*)
1.13: classify_instance(instance,model*)
alt
[class=UP]
1.14: add_to_total_up(MAX(longPF*,0))
[else]
1.15: add_to_total_down(MAX(shortPF*,0))
[else]
1.16: classify_instance(instance,model)
alt
[class=UP]
1.17: add_to_total_up(MAX(longPF,0))
[else]
1.18: add_to_total_down(MAX(shortPF,0))
alt
UP
[totalUP > totalDOWN]
DOWN
[totalUP < totalDOWN]
NONE
[else]

Figure 27. UML sequence diagram for the prediction module's classification task.

3.3 Choosing the Trade Size Using Empirical Knowledge

The prediction module enables the trading agents to automatically decide when to buy or short sell a financial instrument. In order to lower the trading risk, the agents must also be able to decide how much to invest in each trade. More specifically, they should be prepared to use smaller trade sizes when the perceived risk is higher, and to avoid trading when the expected return is negative. The agents that were implemented in the previous section (in consonance with the architecture in Figure 9) completely lacked this skill; they used the same investment amount for all the trades, which is a rather simplistic money management strategy. In an attempt to make the agents a bit more talented, we set out to create a new module that would enable them to select an appropriate size for each transaction. The solution we came up with lets them pick one of three sizes before opening a trade: if the trade is expected to be profitable, a standard, user-defined amount is used; if there are doubts regarding the potential profit of the trade, its size is set to half that amount; finally, if the trade is expected to be unprofitable, the size is set to zero, which in practical terms means that the agent will not open that trade. This mechanism was named "empirical knowledge module", because it uses information from the agent's past trading experience to decide the size of future trades. The module's implementation was accomplished with a case-based reasoning system; in this system, each case corresponds to a trade simulated in a previous period, and contains the following information:

- the price direction forecasted by the prediction module;
- the price direction predicted by each model in the prediction module's ensemble;
- the return that would have been obtained if the forecast of the prediction module was utilized to open a trade in that period.

This mechanism tries to capitalize on the higher profitability associated with certain combinations of the models' predictions. We noticed this correlation when we broke down the trading results of the agents described in the previous section; after analysing the trades they did, it became clear that those trades that were carried out when all the models made the same prediction, i.e., all predicted a price increase or all predicted a price decrease, were usually more profitable than those that were done when the predictions were mixed. Consider the case of the USD/JPY trading agent. Its ensemble has seven models; each of them classifies an instance as either belonging to class *UP* or class *DOWN*, so there are 128 possible combinations for their predictions (2^7). We grouped these different combinations according to the number of *UP* forecasts, and calculated the accuracy and the average return per trade that the agent obtained with the corresponding trades. The results of this analysis can be seen in Figure 28. The chart in this figure confirms that the agent's accuracy was bigger when the models in the ensemble made similar predictions. More importantly, the profit also increased when most of the models made the same classification. In order to make these results even clearer, we grouped the different forecast combinations into two sets, one containing the combinations representing near or total agreement between the models (i.e., all or all but one made the same classification), and the other containing combinations representing greater disagreement between them (i.e., the classification of at least two models differed from all the

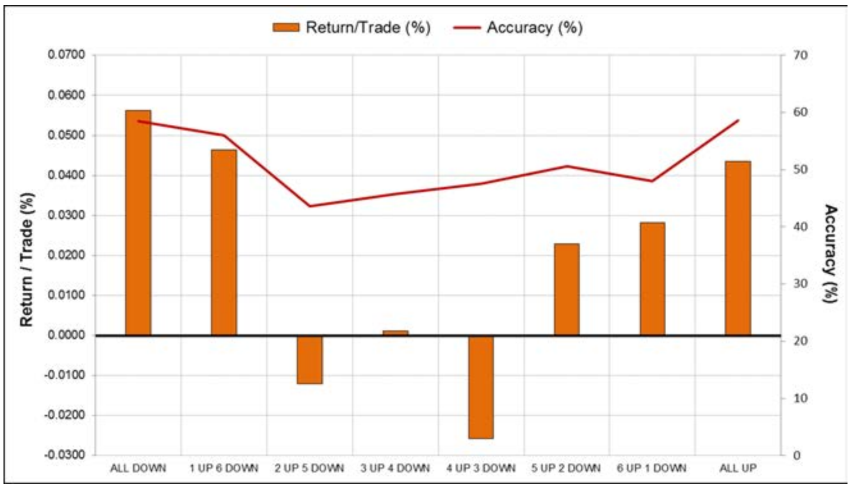


Figure 28. Accuracy and average return per trade of the USD/JPY trading agent for different combinations of models' predictions (excluding trading costs).

others). The statistics of the corresponding trades are synthesized in Table 6; undoubtedly, there is a relationship between the consensus in the models' predictions, and the accuracy and the return of the trades: when all or almost all of the models were in agreement, the classification accuracy of the ensemble was 55.0%; conversely, when the predictions were mixed, that accuracy dropped to 47.8%; the return per trade was also much bigger when the models' classifications were the same (0.0421% versus 0.0009%).

The return and accuracy breakdown for the ADBE trading agent, displayed in Figure 29 and Table 7, show even greater differences. Since this agent's ensemble contains 11 models, the total number of possible combinations is 2,048 (2^{11}). Figure 29 reveals that, during the simulation, there was never a trade for which at least 10 models outputted a *DOWN* classification. Nevertheless, we can still verify that the more homogeneous the predictions, the greater the accuracy and the average return per trade. The results in Table 7 reflect the grouping of the different prediction combinations in two bins: the first represents agreement between the models (at least 9 made the same prediction), and the other represents disagreement between them. On the relatively few times the models agreed, the classification accuracy was 67.2%; that is much higher than the average

Table 6. Accuracy and average return per trade of the USD/JPY trading agent according to the consensus in the models' predictions (excluding trading costs).

Prediction Combinations	*Return/Trade (%)*	*Accuracy (%)*	*Total Trades*
All *UP* 6 *UP*, 1 *DOWN* 1 *UP*, 6 *DOWN* All *DOWN*	0.0421	55.0	1,447
2 *UP*, 5 *DOWN* 3 *UP*, 4 *DOWN* 4 *UP*, 3 *DOWN* 5 *UP*, 2 *DOWN*	0.0009	47.8	928
All Combinations	0.0260	52.2	2,375

accuracy for all combinations (51.0%), and than the 49.6% accuracy when the models disagreed. The average return per trade shows similar discrepancies: 0.3356% when there was agreement, versus just 0.0874% for all combinations, or 0.0666% when there was disagreement.

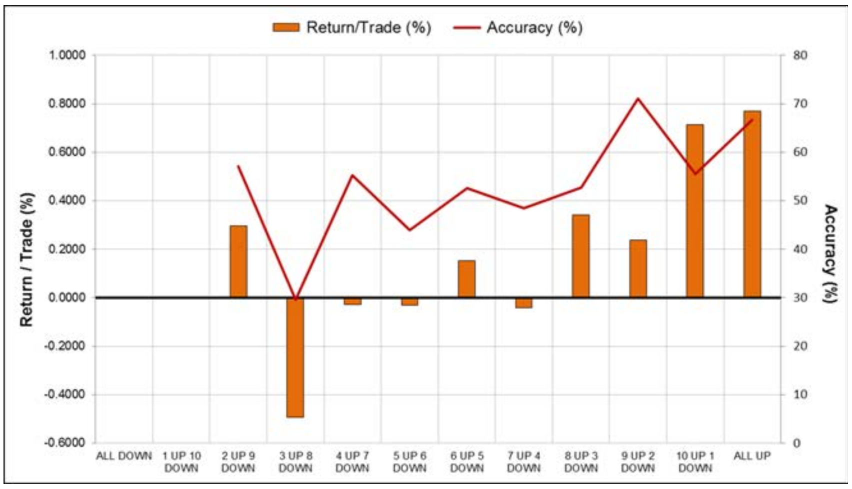


Figure 29. Accuracy and average return per trade of the ADBE trading agent for different combinations of models' predictions (excluding trading costs).

Table 7. Accuracy and average return per trade of the ADBE trading agent according to the consensus in the models' predictions (excluding trading costs).

Prediction Combinations	*Return/Trade (%)*	*Accuracy (%)*	*Total Trades*
All *UP* 10 *UP*, 1 *DOWN* 9 *UP*, 2 *DOWN* 2 *UP*, 9 *DOWN* 1 *UP*, 10 *DOWN* All *DOWN*	0.3356	67.2	64
3 *UP*, 8 *DOWN* 4 *UP*, 7 *DOWN* 5 *UP*, 6 *DOWN* 6 *UP*, 5 *DOWN* 7 *UP*, 4 *DOWN* 8 *UP*, 3 *DOWN*	0.0666	49.6	764
All Combinations	0.0874	51.0	828

Given these empirical results, it is fair to say that the agents might be able to evaluate the profit potential of future trades by looking at the predictions of their models. Thus, they could use this information to decide how much to invest in each trade: if the models make the same prediction, the trade size should be bigger, and if the predictions are mixed, it should be smaller (or even zero). The empirical knowledge module allows them to do just that. We combined this module with the prediction module, and created a new agent architecture, shown in Figure 30. The pseudocode for implementing this architecture is listed in Algorithm 4, and can be visualised in the sequence diagram in Figure 31. This architecture adds a few new steps to the decision process of the trading agent. Before a trade is opened, the prediction module forwards the details of its forecast to the empirical knowledge module; the case-based reasoning system will then retrieve from its database all the cases with the same ensemble prediction and the same combination of models' forecasts; the overall profit factor of the retrieved cases is calculated with Equation 8, and the resulting value is utilized to decide how much to invest in the trade:

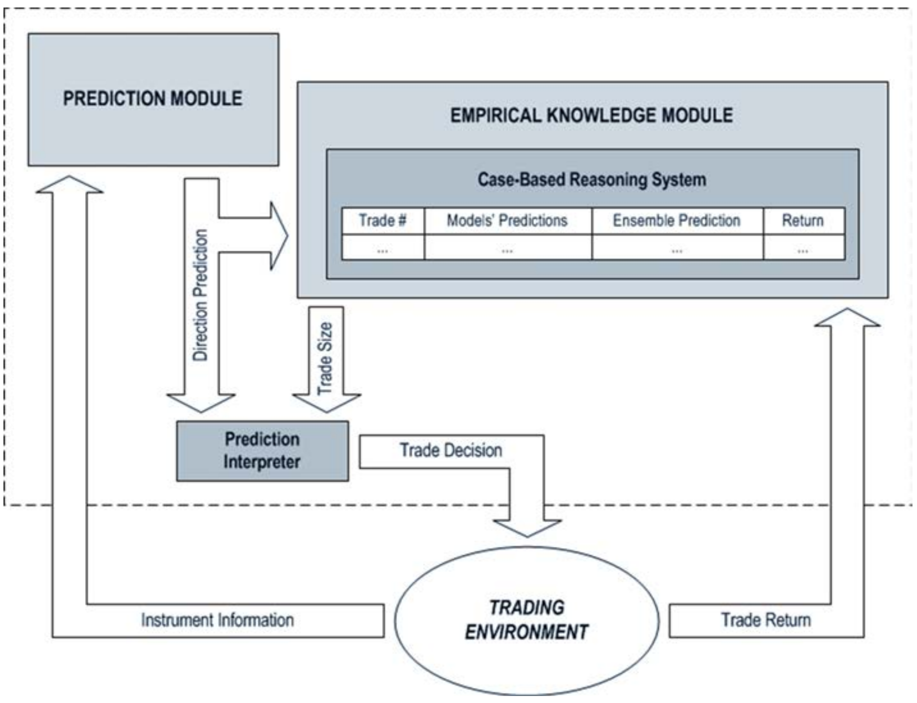


Figure 30. Trading agent architecture based on the prediction and the empirical knowledge modules.

- if the profit factor is less than or equal to a predefined threshold, the trade size is set to zero, and the agent does not trade;
- if it is greater than or equal to another predetermined threshold, the agent invests the standard amount;
- if it is between the two thresholds, the agent invests half the standard amount.

```
Algorithm TradingAgent_v2
Inputs:
    ticker          // instrument to trade
    sAmount         // standard amount to invest in each trade
    ensemble        // ensemble of data mining model that will do the predictions
    N               // test set size
    minCases        // minimum similar cases needed to calculate the profit factor
    minPF           // minimum profit factor to open trade
    highPF          // profit factor threshold to invest the full standard amount

BEGIN
    tradeOpen ← FALSE
    Repeat
        confirmation ← wait_for_end(period)                 // wait for current trading period to end
        If tradeOpen = TRUE Then                            // if a trade was opened in that period:
           confirmation ← close_trade(tradeID)              //     close the trade
           tradeOpen ← FALSE
        EndIf
        periodData ← get_financial_data(ticker,period)                        // get financial data for the period
        confirmation ← add_to_data(ticker,periodData)                         // add period data to the database
        simReturn ← simulate_trade(prevClassPred,periodData)                  // simulate trade with previous prediction
        confirmation ← add_to_cases(ticker,prevClassPred,prevModelPred,simReturn)   // add new case to the CBR database

        // --- PREDICTION MODULE --- //
        class,modelPredictions ← predict_next_class(ticker,ensemble,N)                // Algorithm 3

        // --- EMPIRICAL KNOWLEDGE MODULE --- //
        amount ← trade_size(ticker,class,modelPredictions,sAmount,minCases,minPF,highPF) // suggest trade size (Algorithm 5)

        If class = UP And amount > 0 Then
            tradeID ← buy_instrument(ticker,amount)          // buy if prediction module outputs class UP and empirical
            tradeOpen ← TRUE                                 // knowledge module outputs trade size greater than 0
        ElseIf class = DOWN And amount > 0 Then
            tradeID ← short_instrument(ticker,amount)        // short sell if prediction module outputs class DOWN and
            tradeOpen ← TRUE                                 // empirical knowledge module outputs trade size greater than 0
        EndIf
        prevClassPred ← class
        prevModelPred ← modelPredictions
    EndRepeat
END
```

Algorithm 4. Pseudocode for a trading agent based on the prediction and the empirical knowledge modules.

Note that by following these steps, the agent will find any correlation that might exist between the models' predictions and the profitability of the trades; hence, it is able to ignore the worst models, and to give more importance to the best models, when deciding the trade size. A more detailed description of this decision process is shown in Algorithm 5, and in Figure 32.

We used the new architecture to implement two agents, one to trade the USD/JPY currency pair and the other to trade the ADBE stock. Their cumulative returns during the simulation period are presented in Figures 33 and 34, in comparison with those of the simpler agents that were tested in the previous section; the only difference between these agents is that, while the simpler agents always invest the same amount, the new agents use the empirical knowledge modules to select the

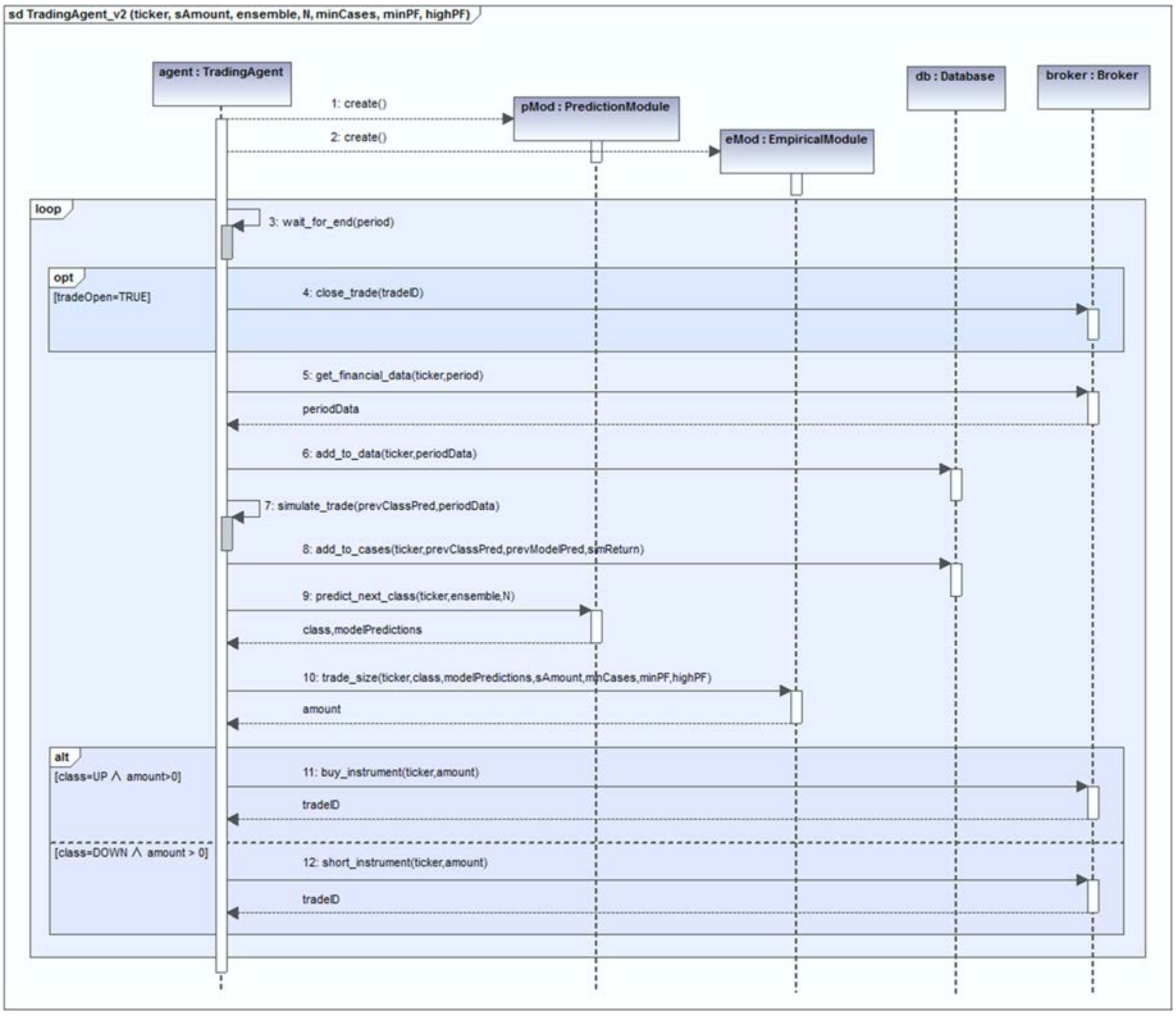

sd TradingAgent_v2 (ticker, sAmount, ensemble, N, minCases, minPF, highPF)
agent : TradingAgent
pMod : PredictionModule
eMod : EmpiricalModule
db : Database
broker : Broker
1: create()
2: create()
loop
3: wait_for_end(period)
opt
[tradeOpen=TRUE]
4: close_trade(tradeID)
5: get_financial_data(ticker,period)
periodData
6: add_to_data(ticker,periodData)
7: simulate_trade(prevClassPred,periodData)
8: add_to_cases(ticker,prevClassPred,prevModelPred,simReturn)
9: predict_next_class(ticker,ensemble,N)
class,modelPredictions
10: trade_size(ticker,class,modelPredictions,sAmount,minCases,minPF,highPF)
amount
alt
[class=UP ∧ amount>0]
11: buy_instrument(ticker,amount)
tradeID
[class=DOWN ∧ amount > 0]
12: short_instrument(ticker,amount)
tradeID

Figure 31. UML sequence diagram for a trading agent based on the prediction and the empirical knowledge modules.

best amount before opening each trade. Therefore, by comparing their results, we can determine how the new module affects the trading performance.

Looking at Figure 33, we verify that adding the empirical knowledge module to the architecture of the USD/JPY trading agent actually decreased its final return. At first glance, this does not seem to be an improvement. However, the smaller return had to be expected. The goal of the empirical knowledge module is to lower the risk, by making the agent skip some trades and put less money on the line when certain conditions are met; this means that the agent takes fewer chances, so it is not surprising that its return is going to be lower, compared to using a fixed trade size – lower risk generally entails lower potential profits. This is not really a problem, because our

```
Algorithm EmpiricalKnowledgeModule_TradeSize
Inputs:
    ticker              // instrument to trade
    class               // ensemble prediction for the next trading period
    modelPredictions    // individual predictions of the models in the ensemble
    sAmount             // standard amount to invest in each trade
    minCases            // minimum similar cases needed to calculate profit factor
    minPF               // minimum profit factor to open trade
    highPF              // profit factor threshold to invest the full standard amount
Outputs:
    amount              // amount to invest in the trade

BEGIN
    totalCases ← 0
    While totalCases < minCases Do                              // while not enough similar cases have been found:
        cases ← retrieve_cases(ticker,class,modelPredictions)   //    retrieve cases with same ensemble and model predictions
        totalCases ← size(cases)                                //    count cases
        modelPredictions ← remove_last(modelPredictions)        //    relax restrictions by ignoring the prediction of the last model
    EndWhile
    returns ← get_returns(cases)                                // get simulated returns of the similar cases
    overallPF ← profit_factor(returns)                          // calculate overall profit factor of the similar cases (Equation 8)
    If overallPF ≥ highPF Then                                  // if the profit factor is greater than or equal to highPF:
        amount ← sAmount                                        //    suggested trade size is the standard amount
    ElseIf overallPF ≤ minPF Then                               // else if the profit factor is less than or equal to minPF:
        amount ← 0                                              //    suggested trade size is 0 (do not trade)
    Else                                                        // else if the profit factor is between minPF and highPF:
        amount ← sAmount / 2                                    //    suggested trade size is half the standard amount
    EndIf
    RETURN amount                                               // output the suggested investment amount
END
```

Algorithm 5. Pseudocode for the empirical knowledge module's trade size decision task.

primary goal is not to develop the most profitable intelligent agents, but rather the agents that can attain the best risk-adjusted return. Proper risk management is one of the few things that distinguishes professional financial trading from ordinary gambling, so we need to make sure that the agent's strategy is as safe as possible – even if that safety comes at the cost of a lower return. If we focus on the risk metrics, the superiority of the new version of the USD/JPY trading agent becomes obvious: its RMD ratio was 5.66, which is better than the 5.35 ratio of the simpler agent. This improvement was possible due to its smaller maximum drawdown, which decreased to 6.3% from 11.5%. These results confirm that the empirical knowledge module worked as expected: it improved the agent's performance, by making its trading strategy less risky. The lower risk is visible in Figure 33; the chart shows very clearly that the cumulative return of the new agent is less volatile than that of the less complex agent. Another factor supporting the usefulness of the empirical

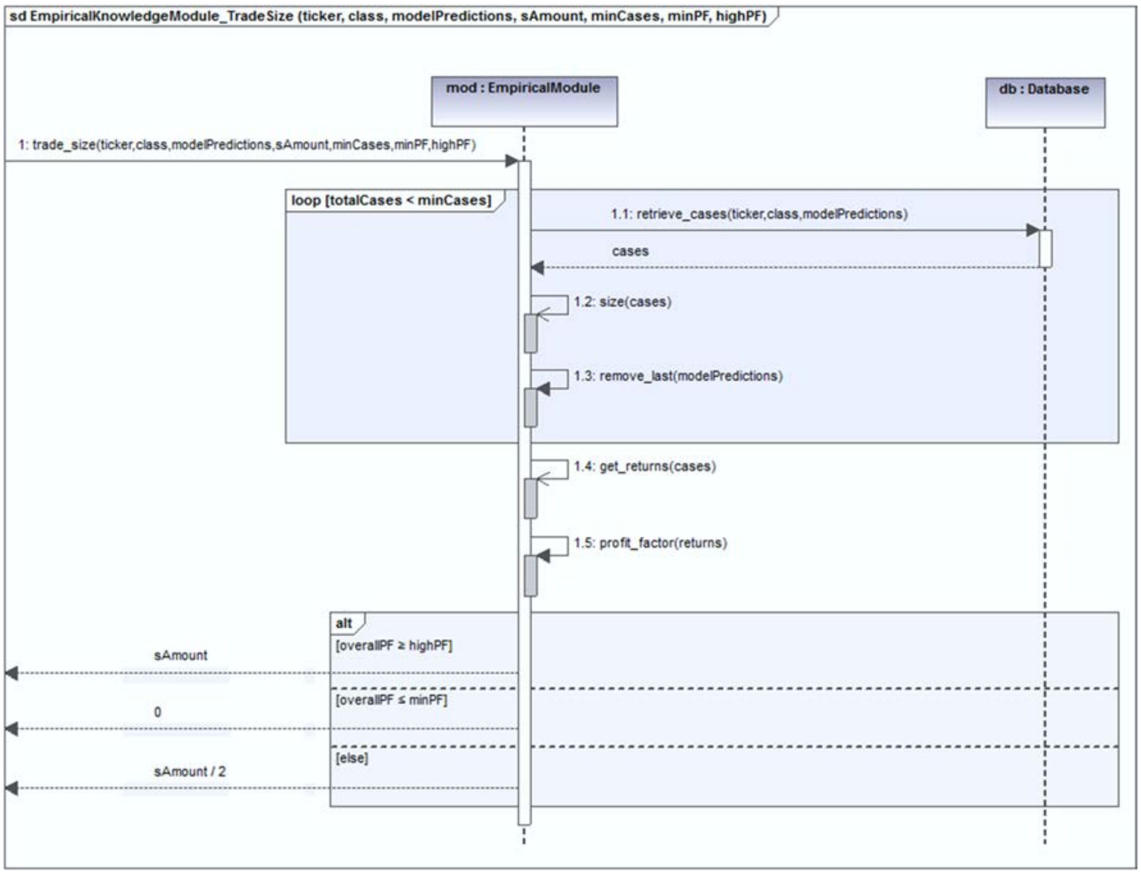

sd EmpiricalKnowledgeModule_TradeSize (ticker, class, modelPredictions, sAmount, minCases, minPF, highPF)
mod : EmpiricalModule
db : Database
1: trade_size(ticker,class,modelPredictions,sAmount,minCases,minPF,highPF)
loop [totalCases < minCases]
1.1: retrieve_cases(ticker,class,modelPredictions)
cases
1.2: size(cases)
1.3: remove_last(modelPredictions)
1.4: get_returns(cases)
1.5: profit_factor(returns)
alt
[overallPF ≥ highPF]
sAmount
[overallPF ≤ minPF]
0
[else]
sAmount / 2

Figure 32. UML sequence diagram for the empirical knowledge module's trade size decision task.

knowledge module was the substantial decrease in the number of trades opened, from 2,375 to just 1,688. This difference implies that, over the course of the simulation, the agent skipped many trades due to its empirical knowledge module predicting they were going to be unprofitable. Still, not all were good news: the return per trade of the new agent dropped to 0.0210% from 0.0260%, which signifies that the decrease in the number of trades was not big enough to compensate for the smaller return.

The empirical knowledge module's ability to lower the risk is more obvious in the results of the ADBE trading agent. We see in Figure 34 that the return curve of the agent that combined the prediction module with the empirical knowledge module is much smoother than that of the agent that only used the prediction module. The simulation statistics confirm the lower risk: the RMD ratio of the more complex agent increased to 2.56 from 1.19, while the return per trade increased to 0.1816% from 0.0874%. The smaller number of trades, 434 instead of 828, was another point in favour of the utility of the empirical knowledge module.

On balance, the results of both agents confirm that the empirical knowledge module can decrease the volatility of the cumulative returns, and in doing so makes their trading strategies less

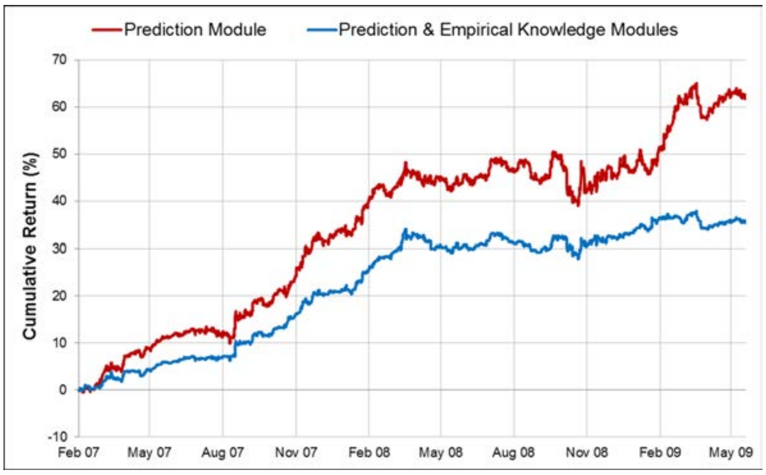


Figure 33. Gross cumulative return of the USD/JPY trading agent based on the combination between the prediction and the empirical knowledge modules.

risky. Moreover, this module brings the agents one step closer to being completely autonomous, by letting them decide for themselves how much to invest in each trade. This leads us to conclude that it could become an important part of the trading agent architecture.

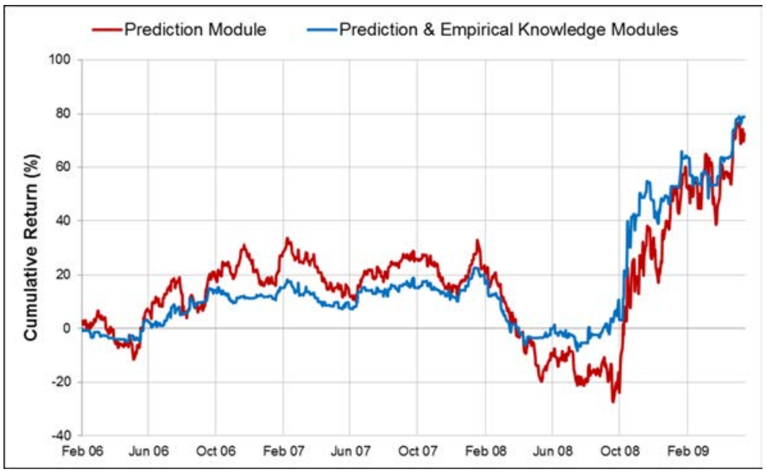


Figure 34. Gross cumulative return of the ADBE trading agent based on the combination between the prediction and the empirical knowledge modules.

3.4 Integrating Domain Knowledge into the Trading Decisions

Both the prediction and the empirical knowledge modules were devised in a way that allows the agents to learn from their empirical trading experiences. But there is always some knowledge that they will not be able to pick up from practice. We created the "domain knowledge module" to overcome this problem. As its name implies, this module's main responsibility is to perfect the trading decisions with domain-specific knowledge. It consists of a rule-based expert system, in which expert human traders insert rules to steer the agents' actions; these rules may pertain to many different aspects of trading – for instance, they can define low liquidity periods when the agents should not trade, or they can compel the agents to close trades if a given profit or loss is reached. These rules will have a very significant impact on the agent's performance. For example, a take-profit rule, i.e., one that forces the agent to close a trade when it reaches a specific profit, can turn trades that would otherwise be unprofitable into successful trades. Whenever a take-profit rule is specified, a trade will be profitable as long as, during the prediction's target period, the price moves in the forecasted direction at least up to the predefined take-profit level; if the price later reverses course, the agent will not be affected, because it will have already closed the trade to lock in the profit. Thus, even if the agent opens a trade based on an inaccurate prediction, it will still be successful in those circumstances. Obviously, the downside of the take-profit rule is that it caps the maximum profit that the agent can get in each trade. A stop-loss rule, on the other hand, has the opposite effect. This rule will close an open trade when the price moves in the "wrong" direction, and the loss reaches a specific amount. Consequently, even if the agent makes an accurate prediction, it will still lose money if the price hits the stop-loss first, and later starts moving in the desired direction. Nevertheless, a stop-loss rule is useful in that it limits how much money the agent stands to lose in each trade, thus preventing it from ever experiencing catastrophic losses.

In order to test the domain knowledge module's contribution to the performance of the trading agents, we designed a new agent architecture, depicted in Figure 35; the pseudocode describing the implementation of this architecture is listed in Algorithm 6, and the corresponding UML sequence diagram is presented in Figure 36. According to this new design, the domain knowledge module is responsible for making the final trading decisions, based on the price direction forecasts (made by the prediction module) and the expert-defined rules. This decision process is detailed in Algorithm 7 and in Figure 37; there is not much to it: the agent asserts facts to the rule engine reflecting updated information about the trading period and the financial instrument, and the rule engine does all the work, by forward chaining the expert rules and outputting the details of the trade to open (among which the price targets to close it). In our implementation, the domain knowledge module was put together with the Drools rule engine[9].

[9] The JBoss Drools rule engine is available at http://jboss.org/drools/.

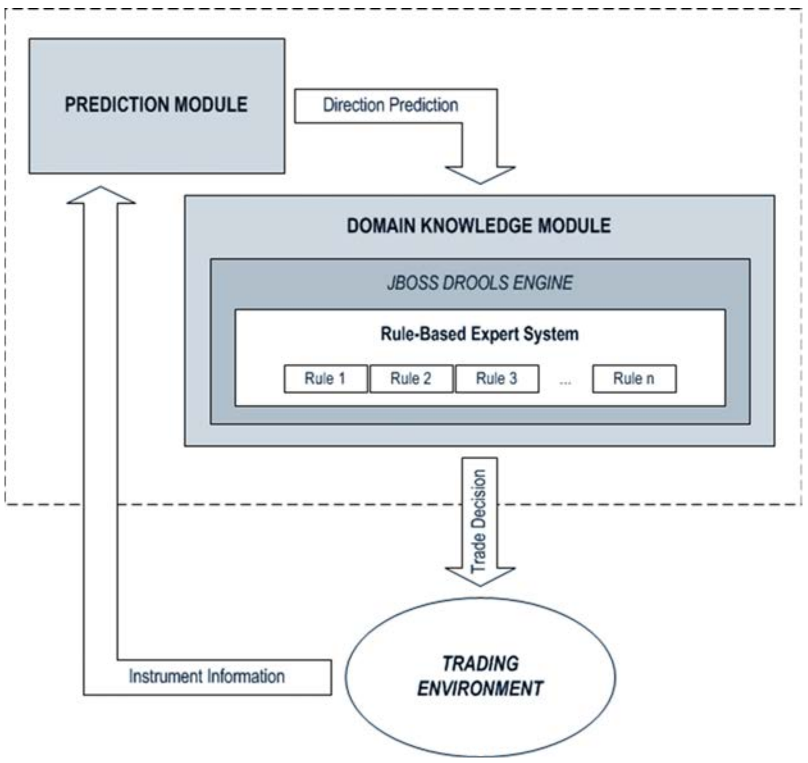


Figure 35. Trading agent architecture based on the prediction and the domain knowledge modules.

Just like we did before, we used the new architecture to implement two agents, one to trade the USD/JPY currency pair and the other to trade the ADBE stock. Their performances over the course of the simulation period are shown in Figures 38 and 39, compared with the performances of the simpler agents that relied solely on the prediction modules (architecture in Figure 9). We should point out that, since the empirical knowledge module was not a part of either design, all the agents employed a fixed trade size throughout the simulation. The first thing we noticed when looking at the results of this experiment was that the cumulative returns of the two new agents were much less erratic than those of the simpler agents. This difference suggests that the domain knowledge module has indeed made them more talented. While the return of the more complex USD/JPY trading agent was smaller than that of the simpler version, its RMD ratio increased from 5.35 to 5.78, and its return per trade improved from 0.0260% to 0.0438%. The much better return per trade was due in part to the lower number of trades, which dropped from 2,375 to 1,205 (less than half of the 2,510 out-of-sample instances). The new ADBE trading agent achieved similar performance gains; its RMD ratio increased from 1.19 to 1.99, while its return per trade increased from 0.0874% to 0.0952%. The number of trades dropped from 828 to 817. Clearly, adding the domain knowledge module to the agents' architecture significantly improved their trading strategies. The rules underlying this improvement will be discussed in Chapters 4 and 5. For now, we can conclude that providing expert domain knowledge to the agents will make them more skilled at avoiding unnecessary risks.

```
Algorithm TradingAgent_v3
Inputs:
    ticker          // instrument to trade
    amount          // amount to invest in each trade
    ensemble        // ensemble of data mining model that will do the predictions
    N               // test set size
    rules           // expert trading rules

BEGIN
    confirmation ← insert_in_engine(ticker,rules)                // add rules to the expert system
    tradeOpen ← FALSE
    Repeat
        confirmation ← wait_for_end(period)                      // wait for the current trading period to end
        If tradeOpen = TRUE Then                                 // if a trade was opened and has not been closed yet:
            confirmation ← close_trade(tradeID)                  //     close the trade
            tradeOpen ← FALSE
        EndIf
        periodData ← get_financial_data(ticker,period)           // get instrument's financial data for the period
        confirmation ← add_to_data(ticker,periodData)            // add period data to the database

        // --- PREDICTION MODULE --- //
        class ← predict_next_class(ticker,ensemble,N)            // Algorithm 3

        // --- DOMAIN KNOWLEDGE MODULE --- //
        class,tp,sl ← make_decision(ticker,class,periodData)     // make final decision according to the expert rules (Algorithm 7)

        If class = UP Then
            tradeID ← buy_instrument(ticker,amount,tp,sl)   // if final decision is UP, buy and send take-profit and stop-loss orders
            tradeOpen ← TRUE
        ElseIf class = DOWN Then
            tradeID ← short_instrument(ticker,amount,tp,sl) // if it is DOWN, short sell and send take-profit and stop-loss orders
            tradeOpen ← TRUE
        EndIf
    EndRepeat
END
```

Algorithm 6. Pseudocode for a trading agent based on the prediction and the domain knowledge modules.

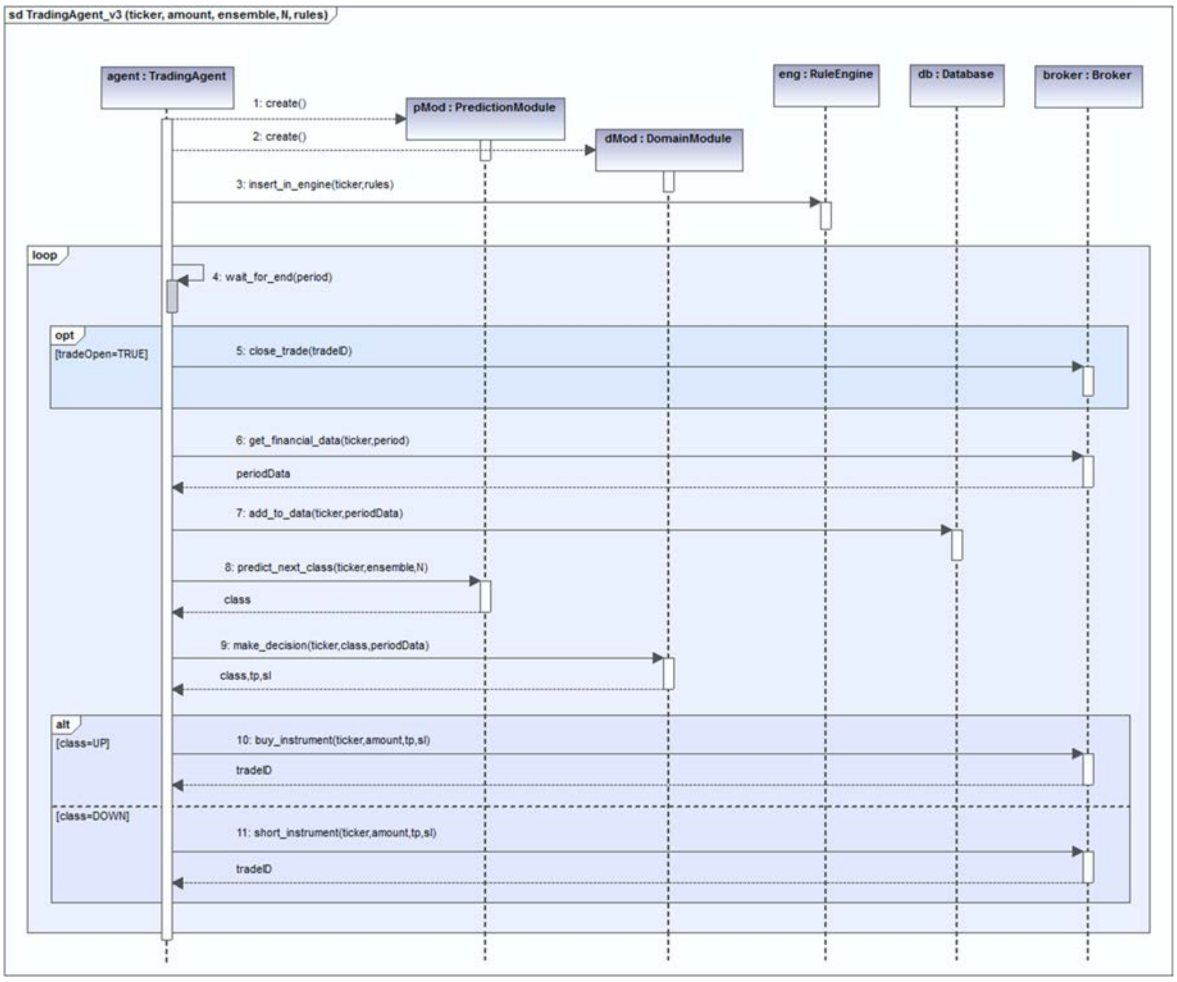

sd TradingAgent_v3 (ticker, amount, ensemble, N, rules)
agent : TradingAgent
pMod : PredictionModule
dMod : DomainModule
eng : RuleEngine
db : Database
broker : Broker
1: create()
2: create()
3: insert_in_engine(ticker,rules)
loop
4: wait_for_end(period)
opt
[tradeOpen=TRUE]
5: close_trade(tradeID)
6: get_financial_data(ticker,period)
periodData
7: add_to_data(ticker,periodData)
8: predict_next_class(ticker,ensemble,N)
class
9: make_decision(ticker,class,periodData)
class,tp,sl
alt
[class=UP]
10: buy_instrument(ticker,amount,tp,sl)
tradeID
[class=DOWN]
11: short_instrument(ticker,amount,tp,sl)
tradeID

Figure 36. UML sequence diagram for a trading agent based on the prediction and the domain knowledge modules.

```
Algorithm DomainKnowledgeModule_MakeDecision
Inputs:
    ticker                  // instrument to trade
    class                   // ensemble prediction for the next trading period
    periodData              // most recent instrument financial data
Outputs:
    class                   // ensemble prediction for the next trading period
    tp,sl                   // orders to close the trade before the period ends if the return reaches certain levels

BEGIN
    confirmation ← assert(ticker,class,periodData)      // assert facts into the rule engine
    class,tp,sl ← fire_all_rules(ticker)                // chain the rules in the engine
    RETURN class,tp,sl                                  // output the final trading decision
END
```

Algorithm 7. Pseudocode for the domain knowledge module's trading decision task.

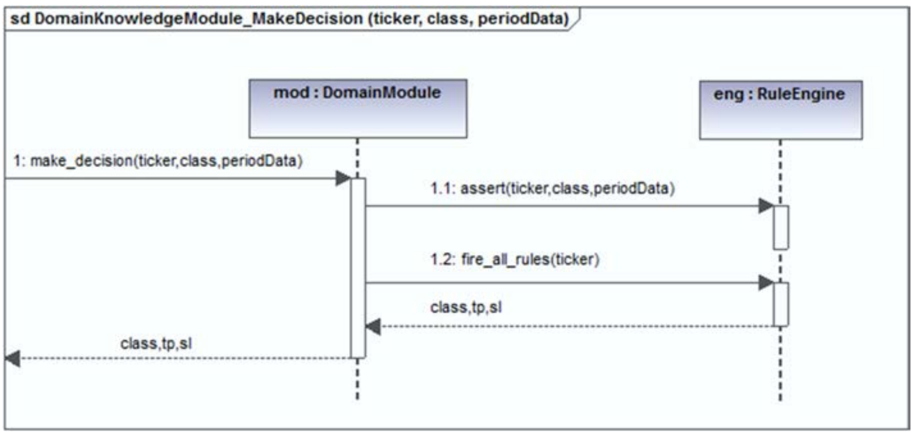

sd DomainKnowledgeModule_MakeDecision (ticker, class, periodData)
mod : DomainModule
eng : RuleEngine
1: make_decision(ticker,class,periodData)
1.1: assert(ticker,class,periodData)
1.2: fire_all_rules(ticker)
class,tp,sl
class,tp,sl

Figure 37. UML sequence diagram for the domain knowledge module's trading decision task.

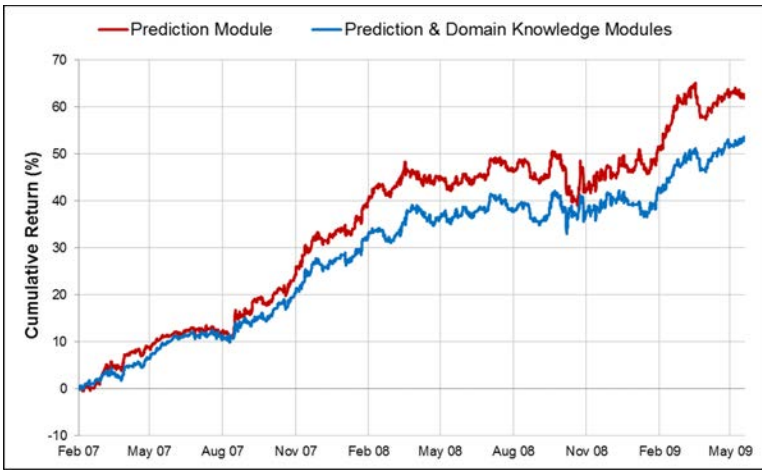


Figure 38. Gross cumulative return of the USD/JPY trading agent based on the combination between the prediction and the domain knowledge modules.

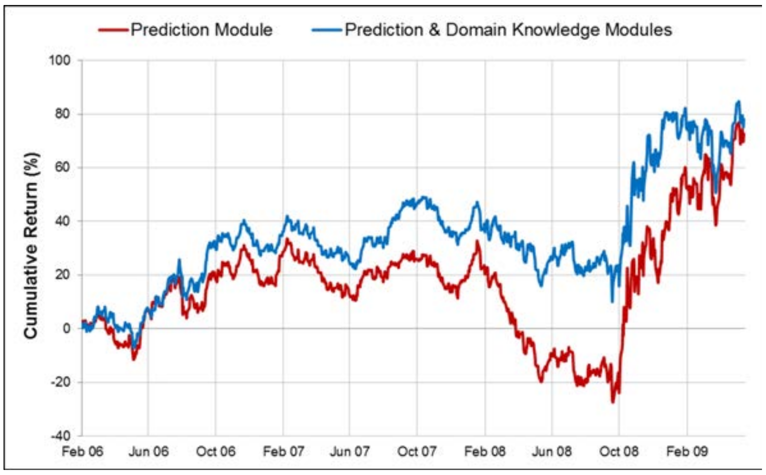


Figure 39. Gross cumulative return of the ADBE trading agent based on the combination between the prediction and the domain knowledge modules.

3.5 The Trading Agent Architecture

In the previous sections, we described three building blocks that could be employed in the development of autonomous trading agents. The first block, the prediction module, is an essential part of the architecture, because it is responsible for making the price direction forecasts that enable the agents to decide when to buy or short sell a financial instrument. We combined this module with each of the other two building blocks (the empirical and the domain knowledge modules), and tested them separately. The results we got demonstrated that, considering the metrics that we deemed most important (the RMD ratio and the return per trade), both modules improved the trading strategy of our agents. We expected agents that could apply both empirical knowledge and expert knowledge in their investment decisions to perform even better. Thus, we combined the three building blocks, and created what constitutes our final proposal for a trading agent architecture. This architecture, shown in Figure 40, is meant to be utilized as the basis for the rapid development of intelligent agents that can trade different types of financial instruments (Barbosa & Belo, 2008a).

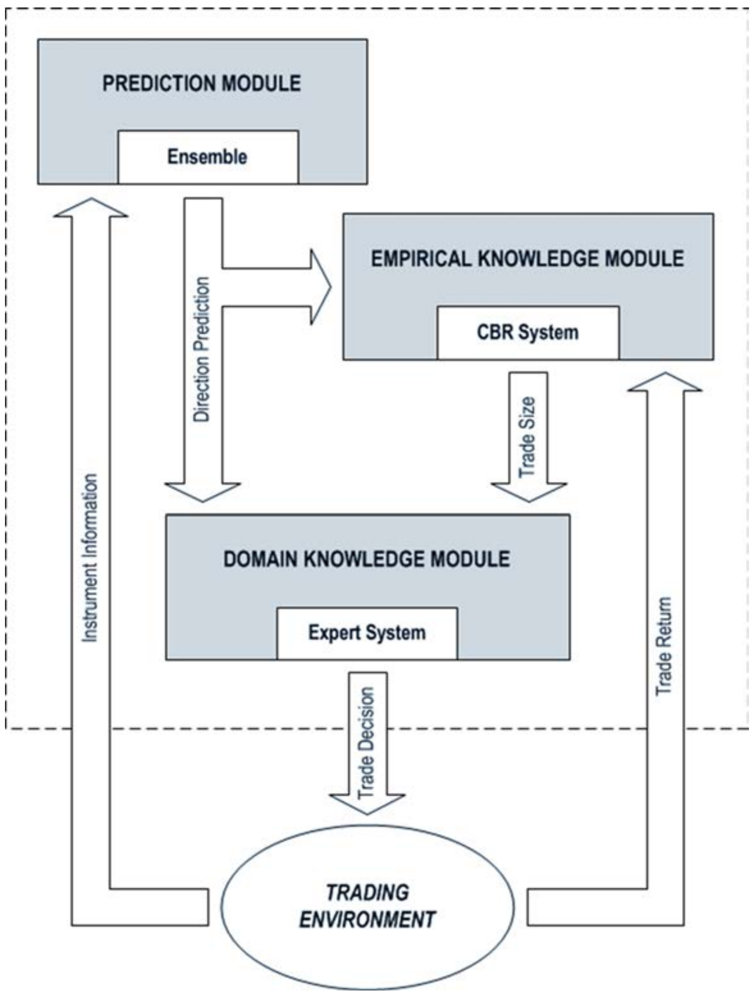


Figure 40. Intelligent trading agent architecture.

Our intelligent trading agent architecture defines the modules' responsibilities as follows:

- the prediction module is responsible for forecasting the direction of the instrument's price; the forecasts are interpreted as suggestions on whether to buy or short sell the financial instrument;
- the empirical knowledge module suggests how much to invest in each trade;
- the domain knowledge module makes the final decisions and opens the trades, in accordance with the suggestions of the other modules and its own expert rules.

The pseudocode that implements this architecture is listed in Algorithm 8; the object interactions are represented in the UML sequence diagram in Figure 41.

Once again, the proposed architecture was used to build two trading agents, one for the USD/JPY currency pair and the other for the ADBE stock. Their cumulative returns in the simulation period are presented in Figures 42 and 43, in comparison with the returns obtained with simpler agent implementations. The results achieved by all the different module combinations that were tested throughout this chapter are summarized in Tables 8 and 9.

Figure 42 shows that, compared to the simpler versions, the USD/JPY intelligent trading agent did not yield as much profit. This is not a problem, because our main concern is capital preservation, not return maximization. In that respect, the intelligent agent outperformed all other implementations, with a RMD ratio of 8.57. Without accounting for trading expenses, its success rate was 56.0%, i.e., 56.0% of the simulated trades were profitable. Its accuracy predicting the direction of the price, on the other hand, was just 53.8%. This disparity was due to the existence of a take-profit rule in the agent's domain knowledge module, that allowed it to secure a profit in trades for which it made the wrong predictions. Out of 2,510 out-of-sample instances, the intelligent agent did only 1,146 trades. Since the accuracy and the success rate increased as we added more modules to the architecture, we can conclude that these modules enabled the agent to avoid several unprofitable trades. Also, we can confirm that each individual module made a positive contribution to its trading ability, and allowed it to be a bit more successful in the end.

```
Algorithm TradingAgent_final
Inputs:
    ticker          // instrument to trade
    sAmount         // standard amount to invest in each trade
    ensemble        // ensemble of data mining model that will do the predictions
    N               // test set size
    minCases        // minimum similar cases needed to calculate the profit factor
    minPF           // minimum profit factor to open trade
    highPF          // profit factor threshold to invest the full standard amount
    rules           // expert trading rules

BEGIN
    confirmation ← insert_in_engine(ticker,rules)          // add rules to the expert system
    tradeOpen ← FALSE
    Repeat
        confirmation ← wait_for_end(period)                // wait for current trading period to end
        If tradeOpen = TRUE Then                           // if a trade was opened and has not been closed yet:
            confirmation ← close_trade(tradeID)            //     close the trade
            tradeOpen ← FALSE
        EndIf
        periodData ← get_financial_data(ticker,period)                         // get financial data for the period
        confirmation ← add_to_data(ticker,periodData)                          // add period data to the database
        simReturn ← simulate_trade(prevClassPred,periodData)                   // simulate trade with previous prediction
        confirmation ← add_to_cases(ticker,prevClassPred,prevModelPred,simReturn)  // add new case to the CBR database

        // --- PREDICTION MODULE --- //
        class,modelPredictions ← predict_next_class(ticker,ensemble,N)                   // Algorithm 3

        // --- EMPIRICAL KNOWLEDGE MODULE --- //
        amount ← trade_size(ticker,class,modelPredictions,sAmount,minCases,minPF,highPF)  // Algorithm 5

        // --- DOMAIN KNOWLEDGE MODULE --- //
        class,amount,tp,sl ← make_decision(ticker,class,amount,periodData)               // Algorithm 7

        If class = UP And amount > 0 Then
            tradeID ← buy_instrument(ticker,amount,tp,sl)      // buy according to the domain knowledge module's decision
            tradeOpen ← TRUE
        ElseIf class = DOWN And amount > 0 Then
            tradeID ← short_instrument(ticker,amount,tp,sl)    // short sell according to the domain knowledge module's decision
            tradeOpen ← TRUE
        EndIf
        prevClassPred ← class
        prevModelPred ← modelPredictions
    EndRepeat
END
```

Algorithm 8. Pseudocode for the intelligent trading agent.

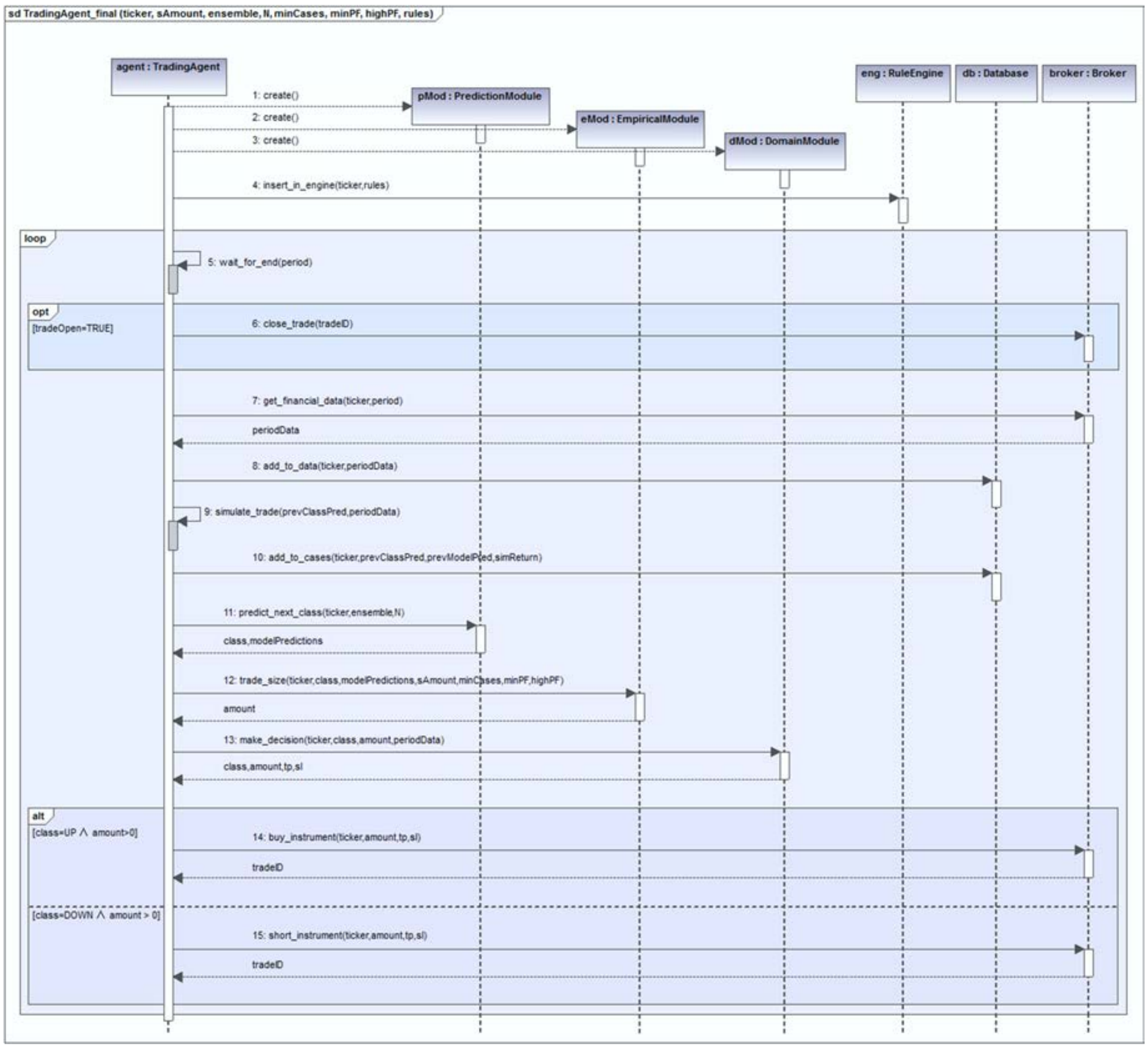

sd TradingAgent_final (ticker, sAmount, ensemble, N, minCases, minPF, highPF, rules)
agent : TradingAgent
pMod : PredictionModule
eMod : EmpiricalModule
dMod : DomainModule
eng : RuleEngine
db : Database
broker : Broker
1: create()
2: create()
3: create()
4: insert_in_engine(ticker,rules)
loop
5: wait_for_end(period)
opt
[tradeOpen=TRUE]
6: close_trade(tradeID)
7: get_financial_data(ticker,period)
periodData
8: add_to_data(ticker,periodData)
9: simulate_trade(prevClassPred,periodData)
10: add_to_cases(ticker,prevClassPred,prevModelPred,simReturn)
11: predict_next_class(ticker,ensemble,N)
class,modelPredictions
12: trade_size(ticker,class,modelPredictions,sAmount,minCases,minPF,highPF)
amount
13: make_decision(ticker,class,amount,periodData)
class,amount,tp,sl
alt
[class=UP ∧ amount>0]
14: buy_instrument(ticker,amount,tp,sl)
tradeID
[class=DOWN ∧ amount > 0]
15: short_instrument(ticker,amount,tp,sl)
tradeID

Figure 41. UML sequence diagram for the intelligent trading agent.

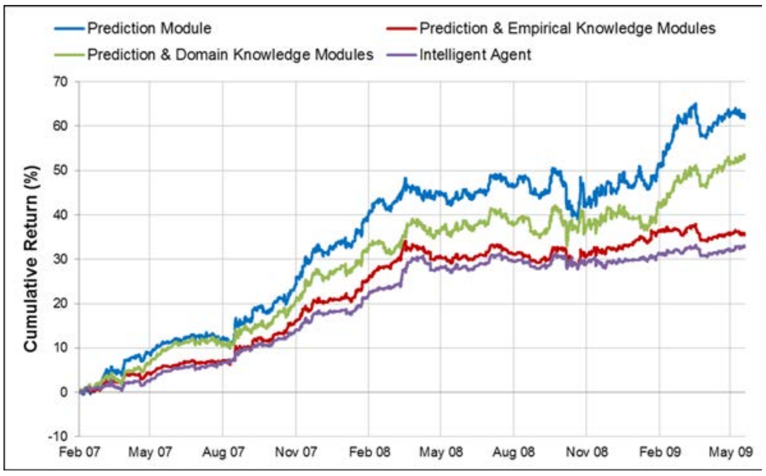


Figure 42. Gross cumulative return of the USD/JPY trading agent based on different architectures.

Table 8. Simulation results of the USD/JPY trading agent using different architectures (excluding trading costs).

Module combination	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Success (%)*	*Trades*
Prediction Module	61.8	11.5	5.35	0.0260	52.2	52.2	2,375
Prediction & Empirical Knowledge Modules	35.4	6.3	5.66	0.0210	53.4	53.4	1,688
Prediction & Domain Knowledge Modules	52.8	9.1	5.78	0.0438	52.4	54.7	1,205
Intelligent Agent	32.7	3.8	8.57	0.0285	53.8	56.0	1,146

The intelligent ADBE trading agent also achieved an interesting performance. Its RMD ratio in the simulation period was 3.20, and the return per trade was 0.1756%; according to these metrics, it did better than the simpler versions, which again confirms the usefulness of each individual module. This intelligent agent opened just 427 trades, out of 828 possible. While its accuracy predicting the daily direction of the stock's price was just 51.5%, it was still able to close 55.0% of the trades with profit. This implies that, even if the behaviour of the price of the financial instrument is extremely hard to predict, the intelligent agent might still be capable of trading profitably – this is an important conclusion, because all our experiments so far have shown that the direction of the price is extremely hard to forecast.

All things considered, we can conclude that both agents performed acceptably in our tests. However, it is possible that this accomplishment was just a fluke, i.e., their success could be due to a simple streak of good luck, which would eventually disappear if they continued trading after the simulation period. There is no way to know for sure if that was really the case, but one can calculate the probability of that happening. In order to do so, we created a "dumb" USD/JPY trading bot that made random decisions on when to buy or short sell the currency pair. The bot was based on the architecture in Figure 4, only we replaced the data mining model with a "coin-flipping" mechanism that made random predictions for the direction of the USD/JPY exchange rate. We

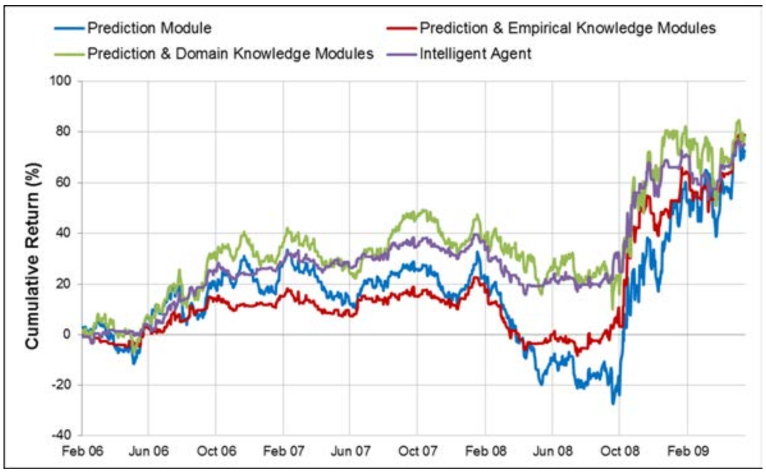
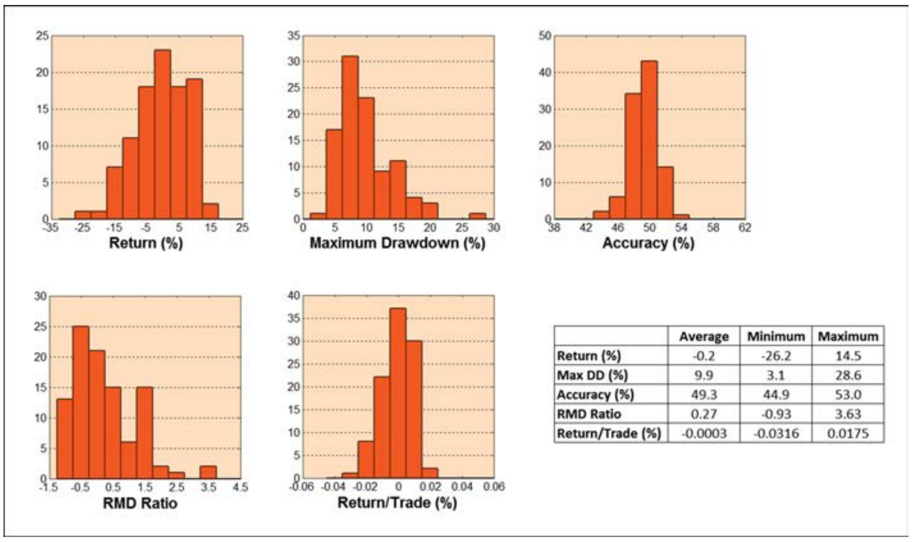


Figure 43. Gross cumulative return of the ADBE trading agent based on different architectures.

Table 9. Simulation results of the ADBE trading agent using different architectures (excluding trading costs).

Module combination	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Success (%)*	*Trades*
Prediction Module	72.4	60.9	1.19	0.0874	51.0	51.0	828
Prediction & Empirical Knowledge Modules	78.8	30.8	2.56	0.1816	51.6	51.6	434
Prediction & Domain Knowledge Modules	77.8	39.1	1.99	0.0952	51.0	53.7	817
Intelligent Agent	75.0	23.4	3.20	0.1756	51.5	55.0	427

used the bot to perform 100 trading simulations, each run consisting of the 2,510 out-of-sample instances that were utilized to evaluate the agents. The histograms in Figure 44 synthesize the bot's performance in these 100 runs. We were expecting the bot's lack of skill to be reflected in its overall performance, and this was clearly the case: the average return for the 100 runs was very close to 0%, and the average accuracy was around 50%. Disregarding the trading costs (which would have had a significant negative impact on the return) this is the type of performance that one should expect from a trading strategy that relies on luck. The bot's best simulation run ended with a return of 14.5% and a RMD ratio of 3.63, which compares poorly with the intelligent agent's 32.7% return and 8.57 RMD ratio. Thus, it is unlikely that the agent achieved that kind of performance just because it was lucky. Nevertheless, even if none of the bot's simulation runs ended with a RMD ratio over 3.6, it is certainly possible that that could occur if we kept repeating the tests. We will use Bayesian statistics to calculate the credible intervals delimiting the probability of that happening. We start with the assumption that there is no prior information regarding the probability of a bot achieving a RMD ratio greater than 3.6, i.e., if we keep repeating the simulation, the proportion of runs that will finish with a RMD ratio over 3.6 may be anywhere between 0 and 1. Hence, the prior distribution for this proportion is uniform. Next, we combine this prior distribution with a binomial distribution summarizing the results of our 100 runs' sample, which yields a beta distribution. Using this posterior distribution, we can calculate credible intervals for the proportion of runs in the population that might finish with a RMD ratio over 3.6. Based on this procedure, we can say with 95% confidence that the probability that a sequence of random predictions throughout the simulation period will result in a RMD ratio over 3.6 is between 0.00025 and 0.03587. This proves that, while possible, it is not very probable that a "dumb" trading bot would ever perform as well as our intelligent agent did. Moreover, we can extrapolate from

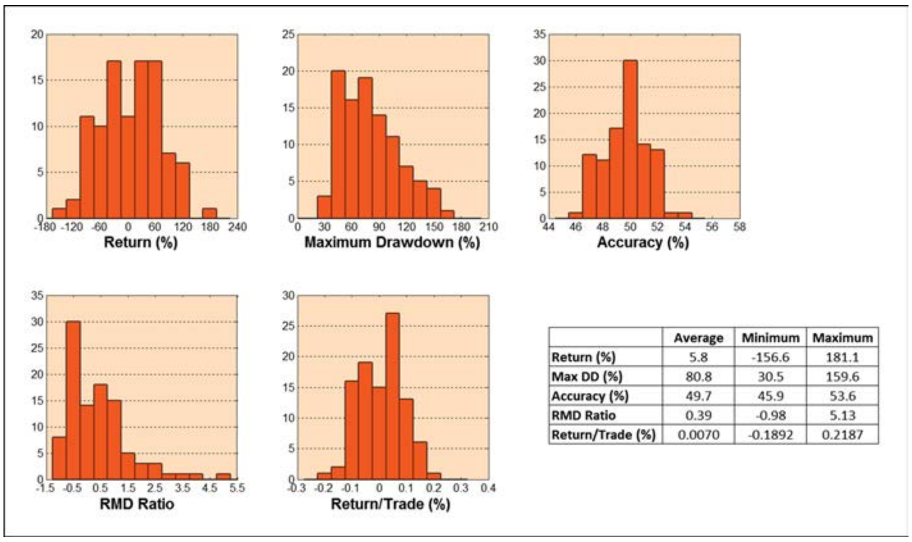


	Average	Minimum	Maximum
Return (%)	-0.2	-26.2	14.5
Max DD (%)	9.9	3.1	28.6
Accuracy (%)	49.3	44.9	53.0
RMD Ratio	0.27	-0.93	3.63
Return/Trade (%)	-0.0003	-0.0316	0.0175

Figure 44. Results of 100 trading simulations with a USD/JPY trading bot that makes random buy and short sell decisions (excluding trading costs).

these credible intervals that it is very unlikely that the USD/JPY trading agent could owe its success to a series of lucky trades.

Following the same strategy, we implemented an ADBE trading bot and tested it in 100 simulation runs; the results of these tests are summarized in Figure 45. The average return in this experiment was 5.8%. This might lead some to believe that, if we were to trade the ADBE stock randomly, we would make a profit more often than not. This cannot be true, or else the majority of traders would be able to trade profitably without much effort. The pitfall here is easy to identify: if we subtract the trading expenses, that average positive return turns into a big loss; so, in real life, the bot would be losing money most of the times. Compared with the agent's 75.0% return and 3.20 RMD ratio, the mean results of the bot were very bad. However, the best run easily surpassed the intelligent agent's performance, with a profit of 181.1% and a ratio of 5.13. While some outliers were to be expected, we still found it mind-boggling that a trading strategy that relied on chance alone could be so successful. This goes to show that, given the many thousands of traders and hedge funds that are currently trying to beat the markets, it is statistically likely that some will end up as huge winners, without their performance conflicting with the efficient market hypothesis.



	Average	Minimum	Maximum
Return (%)	5.8	-156.6	181.1
Max DD (%)	80.8	30.5	159.6
Accuracy (%)	49.7	45.9	53.6
RMD Ratio	0.39	-0.98	5.13
Return/Trade (%)	0.0070	-0.1892	0.2187

Figure 45. Results of 100 trading simulations with an ADBE trading bot that makes random buy and short sell decisions (excluding trading costs).

Obviously, the same reasoning must be applied to our intelligent agents. No matter what, we can never be completely certain that a trader's success is due to skill, and not luck.

As before, we employed Bayesian statistics to calculate the probability of a bot surpassing the RMD ratio of the ADBE trading agent. Out of the 100 simulation runs, three finished with a RMD ratio higher than the agent's. Taking this sample into account, we can say with 95% confidence that, if the simulation is repeated, the probability that the bot will achieve a RMD ratio greater than 3.20 is between 0.01089 and 0.08436. Likewise, since only one run in the sample finished with a return per trade higher than the agent's, the probability that the random investment strategy will outperform the agent is between 0.00241 and 0.05393 for that metric. Neither is very likely, which suggests that the intelligent agent is indeed talented. Nevertheless, we must acknowledge that chance could have played a role in its past success; because of this uncertainty, it would be dangerous to let an agent trade real money based on its historical track record (the maxim "past performance is not a guarantee of future returns" should not be taken lightly).

So far, we have shown that it is statistically improbable that the USD/JPY and the ADBE trading agents could owe their reasonable success in the simulation period to sheer luck. However, this success could still have been just the product of specific market conditions in the test period. If both instruments experienced very low volatility, or their prices continuously trended upward or downward throughout the simulation, it is possible that the agents performed well simply because of those conditions. In other words, the agents' profitability might be correlated with the instruments' price direction, or with the lack of any major negative or positive price spikes in the test data. In either case, if we allowed them to trade real funds and the conditions changed, the results would be disastrous. In order to determine if the two agents were capable of trading profitably regardless of the direction of the price, we plotted their cumulative returns together with the historical prices of the corresponding financial instrument. This is shown in Figures 46 and 47. The USD/JPY price movement in the simulation period is best described as a 2.3 year long bear market, with several bear market rallies in-between. Table 10 shows that the USD/JPY trading agent made most of its profit from short trades; this makes sense, since the price was trending downward most of the time. But the agent also profited from long trades, which indicates its success is not connected to the direction of the price. While some extreme changes in the trend did indeed have a negative impact on its return, the agent was more or less capable of adapting to said changes after some time. As we mentioned before, the negative effect caused by sudden changes in the trend is inevitable, due to there being a delay between the instant the changes occur, and the time they are reflected in the prediction mechanism. The agent might anticipate some of these changes, if it finds the right patterns in the historical data; however, outlier occurrences are almost always completely random and impossible to predict, so the best we can expect is that the agent can react to them, not predict them.

The movement of the ADBE stock price in the 3.3 year long simulation period is harder to describe. It seems somewhat range-bound, with major crashes and rallies occurring from time to time. The ADBE trading agent was able to profit in both situations; for example, it did well between August of 2006 and November of 2007, when the stock price was trending upward, and also performed well after September of 2008, when the price tanked. Most of its profit was

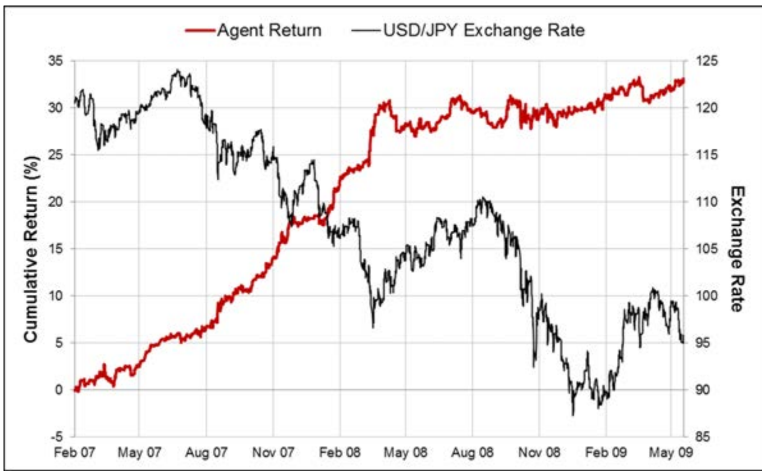


Figure 46. Gross cumulative return of the USD/JPY trading agent versus the USD/JPY price change.

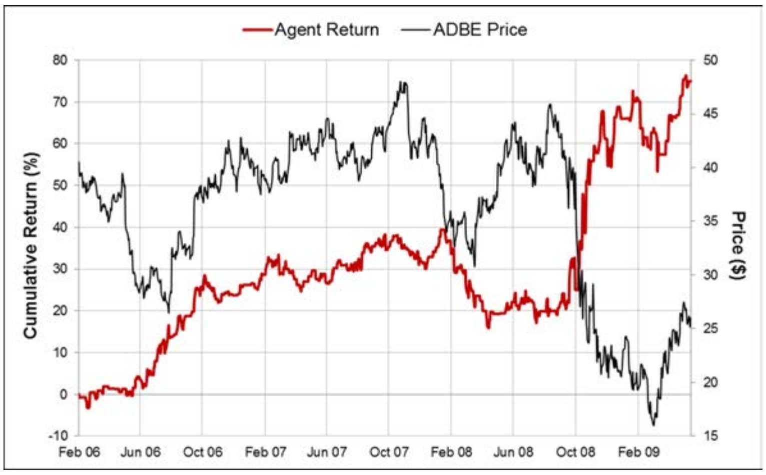


Figure 47. Gross cumulative return of the ADBE trading agent versus the ADBE price change.

Table 10. Return of the USD/JPY and ADBE trading agents according to the type of trades (excluding trading costs).

Agent	*Long Trades Return (%)*	*Short Trades Return (%)*	*Total Return (%)*
USD/JPY	6.5	26.2	32.7
ADBE	19.1	55.9	75.0

obtained with short trades, probably because the biggest percentage price changes occurred when the price was falling. All things considered, it does not look like this agent's success is biased towards a particular price direction. On the flip side, it experienced several significant losses due to sudden increases in the volatility of the price, and it took it almost a year to recover from its biggest drawdown.

From what we have seen so far, either the USD/JPY and the ADBE trading agents were extremely lucky, or they were actually capable of taking advantage of patterns discovered by their data mining mechanisms. If there were no useful patterns in the training data, we would expect them to perform much worse. To test this assumption, we created a random price series for a fictitious financial instrument; a starting price of $100 was gradually altered by adding random uniformly distributed percentage changes (with a small bias towards positive changes, otherwise the price would quickly trend towards zero); the price was altered 6,000 times, each corresponding to a trading period. The resulting price series is displayed in Figure 48. Notice this experiment is similar to Malkiel's (1985), in which he had an expert technical analyst find patterns in a random price series. Since the changes are random, we are certain that there are no patterns in the historical data that can predict future prices. Any trader that tries to forecast the price will probably fail miserably; if successful, we can be sure it was due to luck, not talent. We implemented two new agents to trade this fictitious instrument: agent RAND1, with 7 models in its data mining ensemble, and agent RAND2, with 11 models. They were created using the same method that was utilized with the USD/JPY and the ADBE intelligent agents (which will be described in detail in the next chapter). The first 3,000 periods in the price series were used to train the agents, and the rest was saved for the trading simulations; their cumulative returns in these simulations are shown in Figure 49 and Table 11. The results show that both agents were unable to trade successfully, which simply means that they were not lucky enough; as expected, their accuracy forecasting the random prices was around 50%. This experiment demonstrates very clearly that the architecture we are proposing does not confer the trading agents any "supernatural" powers. They will only perform well if there is useful information in the historical data that is fed to their data mining models. Hence, we can

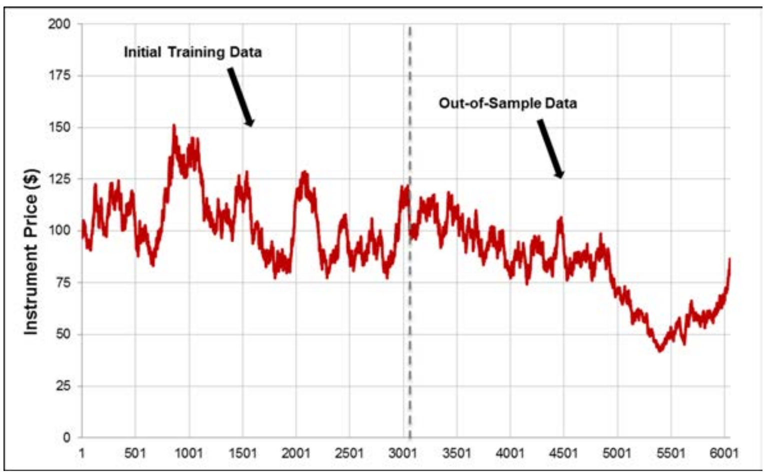


Figure 48. Random price series of a fictitious financial instrument.

conclude that what will ultimately make the difference between a competent and an incompetent trading agent is the choice of training attributes and models in its ensemble.

After comparing the results of the USD/JPY and the ADBE intelligent agents with those of the agents that predicted the random data, we believe it is fair to say that the former do possess some trading talent. Still, just because they did ok in the past, we cannot be 100% certain that they will always be successful in the future. Because of this, we had to come up with an investment strategy that could accommodate for the possibility that an agent might hit a rough patch later on, or that it might turn out to be completely incompetent (which will occur if its previous success was based on luck, and that luck finally runs out). This strategy will be described in Chapter 4; besides providing some resilience to temporary periods of unsuccessful trading by any given agent, it will also serve the purpose of mitigating the impact of volatility spikes and drawdowns in the trading results.

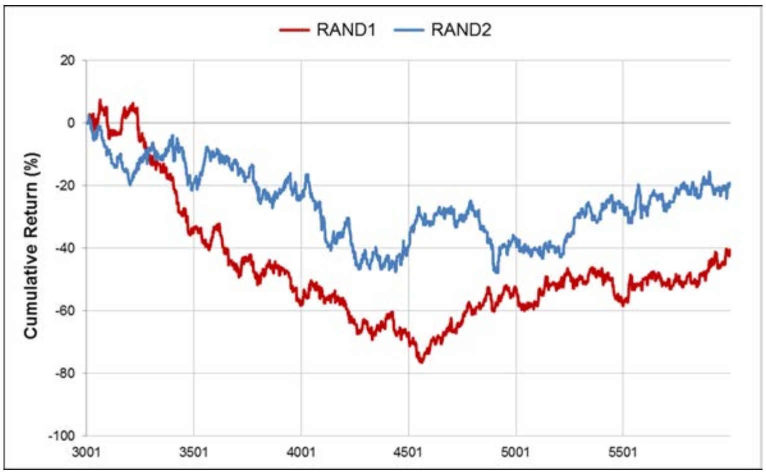


Figure 49. Gross cumulative returns of the RAND1 and RAND2 agents that traded the fictitious instrument.

Table 11. Simulation results of the RAND1 and RAND2 agents that traded the fictitious instrument (excluding trading costs).

Agent	*Ret (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Acc (%)*	*Succ (%)*	*Trades*
RAND1	-40.5	83.9	-0.48	-0.0265	49.6	52.1	1,529
RAND2	-19.2	50.3	-0.38	-0.0142	50.4	52.5	1,350

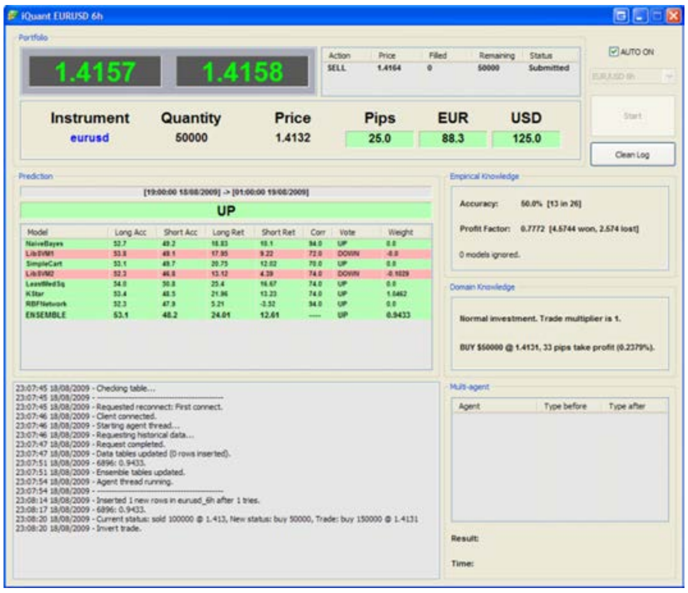
3.6 Streamlining the Implementation of Trading Agents

Both the USD/JPY and the ADBE intelligent trading agents achieved promising simulation results, and in doing so demonstrated the potential of the agent architecture depicted in Figure 40. In order to facilitate the development of new agents, to trade other financial instruments with different time frames, we implemented this architecture as a software "shell". This software was named "iQuant", short for intelligent quantitative analyst; a screenshot is shown in Figure 50. The iQuant software makes it easy to create an intelligent agent to trade any type of financial instrument (stocks, currencies, options, etc.). Specifically, the implementation of a new agent is accomplished by:

- specifying the classes of the data mining models that will compose the ensemble in its prediction module, along with their training parameters and attributes;
- specifying the number of test instances to be used in the calculation of the profit factors of these models;
- setting the two parameters in the empirical knowledge module that determine when the size of a trade should be halved or set to zero;
- specifying the rules in its domain knowledge module.

The only thing that is necessary to implement a trading agent with the iQuant software is historical price data for the instrument it will trade. The agent's trading time frame will depend on the periodicity of that data, and the way it is transformed into training instances.

This software was created using the Java programming language. It utilizes the Weka API (Witten & Frank, 2005) to train and test the data mining models in parallel. The rules in the domain knowledge module are handled by the Drools engine. Interaction with the markets is achieved with the proprietary API of a broker, which allows the agents to send buy, sell, short sell and cover orders automatically. The agents can also be set to perform predictions without



Action	Price	Filled	Remaining	Status
SELL	1.4164	0	50000	Submitted

Instrument	Quantity	Price	Pips	EUR	USD
eurusd	50000	1.4132	25.0	88.3	125.0

[19:00:00 18/08/2009] -> [01:00:00 19/08/2009]

UP

Model	Long Acc	Short Acc	Long Ret	Short Ret	Corr	Vote	Weight
NaiveBayes	52.7	49.2	18.83	10.1	94.0	UP	0.0
LibSVM1	53.8	48.1	17.95	9.22	72.0	DOWN	-0.0
SimpleCart	53.1	49.7	20.75	12.02	70.0	UP	0.0
LibSVM2	52.3	46.8	13.12	4.39	74.0	DOWN	-0.1029
LeastMedSq	54.0	50.8	25.4	16.67	74.0	UP	0.0
KStar	53.4	48.5	21.96	13.23	74.0	UP	1.0462
RBFNetwork	52.3	47.9	5.21	-3.52	94.0	UP	0.0
ENSEMBLE	53.1	48.2	24.01	12.61	—	UP	0.9433

Accuracy: 50.0% [13 in 26]

Profit Factor: 0.7772 [4.5744 won, 2.574 lost]

0 models ignored.

Normal investment. Trade multiplier is 1.

BUY $50000 @ 1.4131, 33 pips take profit (0.2379%).

Agent	Type before	Type after

Result:

Time:

```
23:07:45 18/08/2009 - Checking table...
23:07:45 18/08/2009 - ----------------------------------------
23:07:45 18/08/2009 - Requested reconnect: First connect.
23:07:46 18/08/2009 - Client connected.
23:07:46 18/08/2009 - Starting agent thread...
23:07:46 18/08/2009 - Requesting historical data...
23:07:47 18/08/2009 - Request completed.
23:07:47 18/08/2009 - Data tables updated (0 rows inserted).
23:07:51 18/08/2009 - 6896: 0.9433.
23:07:51 18/08/2009 - Ensemble tables updated.
23:07:54 18/08/2009 - Agent thread running.
23:07:54 18/08/2009 - ----------------------------------------
23:08:14 18/08/2009 - Inserted 1 new rows in eurusd_6h after 1 tries.
23:08:17 18/08/2009 - 6896: 0.9433.
23:08:20 18/08/2009 - Current status: sold 100000 @ 1.413, New status: buy 50000, Trade: buy 150000 @ 1.4131
23:08:20 18/08/2009 - Invert trade.
```

Figure 50. Screenshot of the iQuant software running the EUR/USD trading agent.

submitting trades to the market, meaning they can function as autonomous traders, or as tools to aid other traders. When working autonomously, there are still some situations in which these agents might require human intervention. For example, they cannot be expected to recover from a permanent network disconnection, or from a broker-specific problem. To accommodate for these potential problems, the iQuant software enables the agents to place phone calls: if a critical error occurs, they are able to request assistance by calling the system administrator's cell phone.

In the long run, we expect the agents developed with the iQuant software to demonstrate the same qualities shown by the USD/JPY and the ADBE trading agents. More concretely, considering the agent architecture in which the software is based, we expect the new agents to:

- Keep learning new patterns as time goes by, as a result of the periodic retraining of the models in the ensembles. This process is essential to the agents' autonomy, because it lets them update their prediction mechanisms without requiring external assistance.
- Capitalize on the fact that some models are more profitable under certain market conditions than others. This is accomplished with the continuous reweighting of the models' votes, according to their returns: those that have displayed more profitability in

the recent past will see their vote increase in weight, while those that have been less profitable will lose weight. Since the vote of each model can have two different weights (depending on whether it predicts a price increase or a price decrease), the reweighting mechanism should also enable the agents to take advantage of the fact that some models are better at predicting long trades, while others are better at predicting short trades. This should permit them to adapt to changes in market conditions, as long as some of the models can still perform well under the new conditions.

- Stay out of the market when necessary. This is one of the most important qualities that the agents built with our architecture exhibit: the ability to temporarily stop trading, when that is perceived to be the best option. Trading may be stopped by any of the modules in the architecture: the prediction module will do so whenever all the models in the ensemble demonstrate negative profitability in the recent past, because all their votes will have negative weights; the empirical knowledge module will set a trade's size to zero, and therefore stop it from being made, whenever the cases in its database show that similar trades in the past were unprofitable; finally, the domain knowledge module can restrict the trading activity to certain periods of time or to specific price ranges, among many other conditions that the human experts might define.
- Focus on profit optimization, rather than accuracy optimization. While it is true that the learning algorithms used for retraining the data mining models will optimize their accuracy, the architecture's goal as a whole is to optimize the return: the decision to put retrained models in the ensemble is based entirely on their recent past profitability, as are the models' vote weights; the trade size decisions by the empirical knowledge module are also based on the past profitability of similar cases.

While it is likely that iQuant's generic solution for implementing intelligent trading agents will not work well with all financial instruments and time frames, it at least opens the door to the development of bigger and better systems made up of multiple agents. In these systems, the individual results of each agent are irrelevant – all that matters is the system's performance as a whole, hence a few incompetent agents will not be very problematic. The implementation of this type of system will be the subject of Chapters 4 and 5.

Chapter 4

Intelligent Agents as Autonomous Forex Traders

In the previous chapter we described the step-by-step implementation of two intelligent agents, which traded the USD/JPY currency pair and the ADBE stock. Several details were missing from that description. For instance, we did not state which parameters and attributes were used to train the models in the ensembles; the method with which these models and attributes were selected was also not provided. This information will be presented in this chapter. We will start by briefly describing the algorithms behind numerous data mining models that one can utilize to create the prediction mechanisms of the trading agents, as well as the set of attributes that were defined for their training. Next, we will detail our strategy for selecting the models that compose each agent's prediction module. At that point, we will have clarified the way the agents perform the financial data mining. Then, we will describe how the iQuant software was used, in conjunction with our model selection strategy, to implement ten Forex trading agents, each of which was configured to trade one of the currency pairs listed in Table 12. These ten agents will simulate trades for the same out-of-sample data that was utilized to test the USD/JPY trading agent (corresponding to a period of about 2.3 years) and their results will be discussed later in the chapter. Since there is not guarantee that these agents will be able to trade successfully going forward, we will attempt to diminish the risk associated with their individual strategies by devising a safer diversified

Table 12. Description of the currency pairs traded by the Forex agents.

Pair	*Description*
CHF/JPY	Price of 1 Swiss franc in Japanese yen.
EUR/CHF	Price of 1 Euro in Swiss francs.
EUR/GBP	Price of 1 Euro in British pounds.
EUR/JPY	Price of 1 Euro in Japanese Yen.
EUR/USD	Price of 1 Euro in U.S. dollars.
GBP/CHF	Price of 1 British pound in Swiss francs.
GBP/JPY	Price of 1 British pound in Japanese yen.
GBP/USD	Price of 1 British pound in U.S. dollars.
USD/CHF	Price of 1 U.S. dollar in Swiss francs.
USD/JPY	Price of 1 U.S. dollar in Japanese yen.

investment strategy that employs them all simultaneously. Lastly, we will describe the integration of the ten agents in a multi-agent system, with the objective of improving the performance of the diversified strategy.

4.1 Data Mining Algorithms

Data mining is the name given to the process of extracting hidden patterns from large amounts of data (Fayyad *et al.*, 1996). In order to train a predictive data mining model, the raw data must first be converted into a set of instances; an instance is a vector containing the values of a group of attributes or features (the independent variables) in conjunction with the value of its class (the dependent variable). During the training phase, the data mining algorithms analyse the instances, and try to identify combinations of attribute values that can accurately predict the nominal label (in classification problems) or the numeric value (in regression problems) of the class, a process that is known as supervised learning. After training, the resulting models can be utilized to forecast the class value of out-of-sample instances, i.e., instances that were not part of the training set.

Prediction models come in many flavours – decision trees, rule learners, artificial neural networks, etc. Each type is characterised by two distinguishing algorithms: the training algorithm, which defines the way the model attempts to find patterns in the training instances (i.e., the way it learns), and the classification algorithm, which determines how those patterns are used to predict the class value of new instances. The trading agent architecture proposed in the previous chapter relies on ensembles of models of different types to make financial forecasts. Before creating these ensembles, we believe it is important to understand how the different models learn and generalize from the training data. Therefore, we will describe very briefly the algorithms underlying all the models that we intend to utilize later on. Please note that the extent of each description will reflect not only the importance and complexity of the model, but also the clarity of the information available on it. For the more obscure algorithms, only a small description will be provided.

4.1.1 Instance-Based Models

Instance-based classifiers are data mining models that classify out-of-sample instances by finding the training instances that are similar to them, checking their class values, and then outputting the most frequent class. They are lazy models, because all they do during the training phase is store the instances for later use; generalization from these instances is delayed until the model is faced with an out-of-sample instance to classify.

Arguably the most well-known instance-based classifier is the k-nearest neighbour model (Aha *et al.*, 1991), which treats the instances as points in the feature space (an abstract n-dimensional space where instances with n attributes are represented as points, and where the coordinates of an instance are its attribute values). To classify a new instance, the k-nearest neighbour model puts it in the feature space, and locates the k nearest training instances; given the classes of these instances, the class prediction for the new instance is decided by simple majority voting. Several functions can be employed to calculate the distance between two instances in the feature space. Euclidean distance is a common choice; for instances x and y with n attributes, this distance is equal to:

$$\sqrt{\left(a_1^{(x)} - a_1^{(y)}\right)^2 + \left(a_2^{(x)} - a_2^{(y)}\right)^2 + \cdots + \left(a_n^{(x)} - a_n^{(y)}\right)^2}$$

where $a_n^{(x)}$ and $a_n^{(y)}$ represent the values of the nth attribute of instances x and y, respectively. Since the square root operation is redundant for the purpose of finding the closest instances, it is not computed during classification. Note that if we used this formula directly, without pre-processing the data, the numeric attributes with the largest scales of measurement would make the biggest contributions to the distance, meaning they would be much more important to the classification than any other attributes. To avoid this pitfall, it is common practice to normalize all the values before calculating the Euclidean distance, so that they always lie between 0 and 1. This is accomplished by calculating the normalized value a_i for each numeric attribute, which can be done for example with the formula:

$$a_i = \frac{v_i - min(v)}{max(v) - min(v)}$$

where v_i is the actual value of attribute i, and $max(v)$ and $min(v)$ are the maximum and the minimum values for this attribute in the training set. Besides numeric attributes, *k*-nearest neighbour models also support nominal attributes: a distance of 1 is plugged into the equation when the instances have different labels for the attribute, otherwise 0 is used. Missing values are handled similarly, by assuming a distance of 1 for attributes whose value is missing in at least one of the instances. In addition to classification, these models can also be applied in regression problems: instead of returning the most frequent class label among the *k* nearest neighbours, they return the average or the median class value of the neighbours (among other alternatives).

Searching for the closest neighbours of a test instance implies calculating the distance between that instance and all the training instances. This will be prohibitively slow when the training set is very large. In order to overcome this problem, one could employ a nearest neighbour search algorithm to partition the feature space (Bentley, 1980; Friedman *et al.*, 1977). This algorithm creates a data structure, like a ball tree or a kd-tree, that speeds up the classification task because the model will only need to calculate the distances to the training instances located in partitions close to the test instance. The classifier can also be made faster by eliminating redundant instances from the training set, a strategy for which several algorithms have been proposed (Aha, 1992; Wilson & Martinez, 2000).

When configuring a *k*-nearest neighbour model, it is very important to pick an appropriate value for parameter *k*, as this will have a big impact on its accuracy. For example, consider the two-dimensional feature space represented in Figure 51. We placed 11 training instances in this space, according to their values for attributes a_1 and a_2; of these, 6 belong to class *WHITE*, while 5 belong to class *BLACK*. Suppose we are given an out-of-sample instance to classify (the grey dot in the figure). If we use a nearest neighbour model with *k* set to 3, this instance will be classified as belonging to class *BLACK*, because out of its three closest neighbours, two belong to that class; however, if we set *k* to 5, the instance will be classified as belonging to class *WHITE*. So, even though the two models were trained with the exact same training instances, they will output a different classification for the same out-of-sample instance. This example demonstrates why it is so

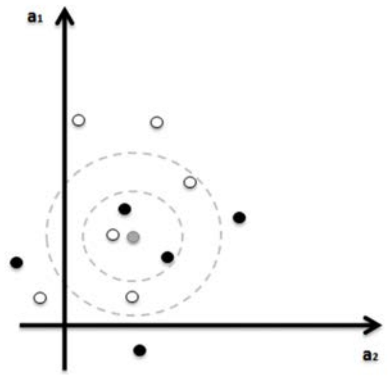


Figure 51. Classification of an out-of-sample instance using two k-nearest neighbour classifiers, with k=3 and k=5.

important to understand the models' training parameters. In practice, larger values of *k* should be utilized when the training data is noisy, but this could result in less distinct boundaries between the classes. One way to select a good value for *k* is to perform cross-validation; this technique implies repeatedly partitioning the training data into two datasets, using one to train the model and the other to test it (Kohavi, 1995a). After performing cross-validation with different *k* settings, the one that yields the best average accuracy is picked for the actual model that will classify the out-of-sample instances.

Besides Euclidean distance, there are other less trivial strategies to determine the similarity between two instances. The K* data mining model (Cleary & Trigg, 1995) uses an entropy-based distance function motivated by information theory. This function defines the distance between two instances as the complexity of transforming one instance into the other, using a sequence of predefined elementary operations.

As part of our research, we came up with our own instance-based model. We named it similarity classifier[10] (Barbosa & Belo, 2009b). Since this is a lazy classifier, the construction of the model (i.e., the learning) is accomplished by simply storing the training instances in an easy to access data structure. Given a new instance of an unknown class, its classification is accomplished by finding all the training instances that are similar to it, and counting the number of times each class occurs in that set. The most frequent class is chosen as the model's class prediction for the new instance.

Our classifier's strategy to decide if two instances are similar is best explained with an example. Imagine a classification problem where each instance is composed of 10 nominal attributes. If these attributes are equally important in determining the class of the instances, we could define that two instances are similar if, for example, at least 5 of their attributes have the same value. Thus, to classify a new instance, the similarity classifier would just need to find all the training instances

[10] The Weka version of the similarity classifier is downloadable at http://ruibarbosa.eu/classifiers/.

with at least 5 attribute values in common with the test instance, count the number of times each class occurs in the set of similar instances, and then choose the most frequent class. However, the assumption that all the attributes have the same importance when predicting the class is incorrect for most real life problems. More often than not, some attributes will be extremely important, while others will be close to irrelevant. Data mining models that do not address this issue are more sensitive to the presence of redundant or useless attributes in the training set – they put the burden of attribute selection completely on the user. That is not the case with the similarity classifier: this model is able to give more relevance to the most important attributes by assigning a different weight to each attribute, proportional to its importance in determining the instances' class. More concretely, the weight of each attribute is given by the value of its correlation with the class feature, calculated with a heuristic devised by Hall (1999). This should be helpful, since attribute weighting has already been successfully applied to several types of data mining models (Kohavi *et al.*, 1997; Hall, 2006). Using these weights, the classification of an out-of-sample instance becomes more complex. First, a threshold is defined that marks the value above which a training instance is considered non-similar to the out-of-sample instance. This threshold is given by:

$$threshold = maxDiff \times \sum_{i=1}^{n} w_i$$

where n is the number of attributes, w_i is the weight of the i^{th} attribute (i.e., the absolute value of the correlation between attribute i and the class in the training set), and $maxDiff$ is a user-defined parameter. In order to decide if two instances are similar, the classifier calculates the difference between them, by adding the weights of the attributes that have different values:

$$difference = \sum_{i\ is\ different} w_i$$

The instances will only be similar if the $difference$ is less than or equal to the $threshold$. This makes the similarity classifier much "smarter". Let us go back to our previous scenario, where the instances had 10 attributes. Assume that the first attribute is a very good predictor of the class, while the other 9 are not. If each attribute is assigned an equal weight of 1, and the user sets $maxDiff$ to 0.5, the similarity classifier will consider two instances similar if at least 5 of their attributes have the same values. This means that, even if the first attribute is different, a training instance will still be found similar to the out-of-sample instance if enough irrelevant attributes are the same. This instance will end up being used in the prediction the class of the out-of-sample instance, even though there is no relationship between the values of the matching attributes and the class. If the classifier uses correlation-based weights instead, the weight of the first attribute will be much bigger than that of the other 9 attributes. Depending on how the user sets the $maxDiff$ parameter, it is possible that two instances will only be considered similar if the value of the first attribute is the same; in other words, if the first attribute is different, the instances will not match even if they share the same values for the other 9 attributes – in practice, this is equivalent to excluding the 9 irrelevant attributes from the classification process.

The interpretation of the $maxDiff$ user parameter is straightforward. It may be set to any value between 0 and 1, and defines the percentage of the total sum of weights above which two instances are treated as non-similar. If it is set to 0, only the training instances that have exactly the same attribute values as the test instance will be found similar to it (because the $threshold$ will be zero, so the $difference$ must be zero too). If it is set to 1, all the instances will match, so the model will always classify new instances as belonging to the most frequent class in the training set.

While the comparison of nominal values is simple, numeric attributes require special treatment. We cannot simply check if two continuous values are equal because, in practical terms, it is very unlikely that they will be. The similarity classifier solves this problem by discretizing all the numeric attributes when the model is created; out-of-sample instances are also discretized prior to being classified. Discretization can be supervised (Fayyad & Irani, 1993) or unsupervised by simple binning. Once discretized, the numeric attributes are treated like regular nominal attributes. The similarity classifier is also capable of handling missing values: when comparing two instances, if the value of an attribute is missing in at least one of them, the attribute is considered different, which means its weight will increase the $difference$ between the instances, making it less likely that they will match.

As previously stated, after finding all the training instances that are similar to a given test instance (by calculating the $difference$ between each of them and the test instance, and checking if the result is less than or equal to the $threshold$), and after counting the number of times each class occurs in the computed set, the similarity classifier will pick the most frequent class as its prediction for the test instance's class; if there is a draw between several classes, it chooses the one with the biggest prior probability. Algorithm 9 shows a high-level description of the methods that characterise the similarity classifier.

Using their default parameters, we tested the three lazy models discussed in this section with various datasets. We verified that the K* and the *k*-nearest neighbour were, in general, slightly more accurate than the similarity classifier. However, the similarity classifier was several orders of magnitude faster than either of them, when classifying out-of-sample instances. It is important to

```
Method BuildModel
Inputs: training instances, $maxDiff$
    Discretize the numeric attributes in the training instances
    Calculate the prior probability of each class
    Assign a weight to each attribute (equal or correlation-based)
    Calculate the $threshold$
    Eliminate instances that do not help distinguish the classes
    Put the instances in an easy-to-access structure

Method ClassifyInstance
Input: test instance
Output: class prediction
    Discretize the numeric attributes in the test instance
    Find all the training instances that are similar to the test instance (those with $difference \leq threshold$)
    Count the number of times each class occurs in the resulting set of instances
    If there is a draw regarding the most frequent class:
        return the one with the biggest prior probability
    Else:
        return the most frequent class
```

Algorithm 9. Pseudocode for the similarity classifier.

recognize these differences, because in some data mining problems the best solution might depend not only on the model's accuracy, but also its speed.

By now, it should be clear that the main advantage that lazy models possess is that their training is extremely fast, because almost nothing is done when they are created. However, instance classification will be very slow, compared to eager models. Another disadvantage that some of them have, and which is shared by other types of models, is that they cannot find patterns that combine the values of different attributes – since they process the attributes separately, one at a time, they are not able to find inter-attribute relationships that might be correlated with the class.

4.1.2 Statistical Regression Models

Statistical regression has been used for decades to create models that fit empirical data. The simplest of these models is the linear regression (Montgomery & Peck, 1982). Its aim is to define an equation that predicts the value of a numeric class c (the dependent variable) given the values of a set of attributes $a_1, \ldots, a_n$ (the independent variables):

$$c = w_0 + w_1 \times a_1 + w_2 \times a_2 + \cdots + w_n \times a_n$$

The weights $w_0, \ldots, w_n$ are calculated from the training data using one of several algorithms. The most common is the ordinary least squares method; considering the predicted class value for training instance j is given by:

$$w_0 + \sum_{i=1}^{n} \left(w_i \times a_i^{(j)} \right)$$

the ordinary least squares procedure will calculate the coefficients $w_0, \ldots, w_n$ by minimizing:

$$\sum_{j=1}^{t} \left(y^{(j)} - \left(w_0 + \sum_{i=1}^{n} \left(w_i \times a_i^{(j)} \right) \right) \right)^2$$

where t is the total number of training instances, and the expression inside the parentheses represents the difference between the actual numeric class value $y^{(j)}$ of training instance j and the predicted value for that instance (this difference is known as the j^{th} residual). Once the weights have been calculated, the model can be utilized to predict the class value of out-of-sample instances by plugging their attribute values into the equation.

The presence of outliers in the training data has a very negative impact on the predictive ability of linear regression models, because the ordinary least squares method is too sensitive to the outliers (they cause big residuals that skew the coefficients). The least median squared linear regression (Rousseeuw & Leroy, 1987) is more robust, because it is less affected by those extreme observations. The construction of this model is accomplished by applying standard linear regression to random subsamples of the training data, and picking the linear regression with the smallest median of squared residuals as the final model.

The pace regression model (Wang & Witten, 2002) is another take on linear regression. To train it, we must select one of several estimators (PACE1, PACE2, …, PACE6). These estimators offer different ways to calculate the weights of the linear model; some include tricks to improve it, like discarding attributes that are found to be redundant.

The statistical regression models referenced so far are all intended for data mining problems with numeric classes. The logistic regression model (Cessie & Houwelingen, 1992), on the other hand, is meant for nominal classes. Given a binary classification problem with class labels c_1 and c_2, it calculates the probability of an instance with attributes $a_1, \dots, a_n$ belonging to class c_1 using the logistic function:

$$P(c_1|a_1, \dots, a_n) = \frac{1}{1 + e^{-(w_0 + w_1 \times a_1 + w_2 \times a_2 + \cdots + w_n \times a_n)}}$$

The probability that it belongs to class c_2 is then easily determined with:

$$P(c_2|a_1, \dots, a_n) = 1 - P(c_1|a_1, \dots, a_n)$$

The weight w_0 is called intercept, while weights $w_1, \dots, w_n$ are called regression coefficients. Several algorithms are available to calculate these weights, among which the iterative reweighted least-squares method. Attributes with positive regression coefficients increase the probability that the instance belongs to class c_1, while attributes with negative coefficients decrease that probability. Besides binary classes, the logistic regression model can also address multiclass problems by way of pairwise classification, which implies training one classifier for each pair of classes – to predict the class of an out-of-sample instance, all the models will classify it, and the class is chosen by majority vote.

4.1.3 Tree Inducers

Some of the most important data mining models are internally structured as trees. The C4.5 decision tree (Quinlan, 1993) is one such model. Each leaf in this tree represents a classification, while the branches that connect the leaf to the root node equate to conjunctions of conditions that lead up to that classification. Figure 52 shows a sample C4.5 decision tree (for a binary classification problem); note that a_2 is a numeric attribute, while a_1 and a_3 are nominal attributes, with possible values *x*, *y*, *z*, and *t*, *u*, respectively.

The C4.5 algorithm makes use of the concept of entropy, as defined by information theory, to grow the tree from a set of training instances. The tree is constructed iteratively, one node at a time, using a divide-and-conquer strategy – each node represents a split in the data that separates the training instances into different branches, according to the values of a specific attribute. In each

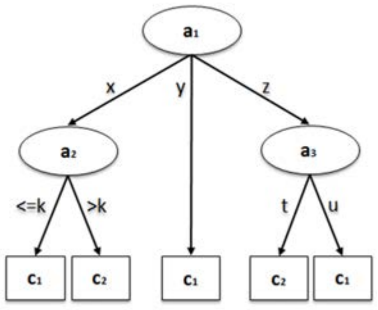


Figure 52. Graphical representation of a C4.5 decision tree.

iteration, the following sequence of steps is applied to each attribute, in order to decide which one should be used to split the data:

- Use the attribute to split the instances. If the attribute is nominal, this is accomplished by creating a branch for each of its possible labels. If it is numeric, a pair of branches is created using a threshold: one branch for the instances with an attribute value greater than the threshold, and the other for the rest. The threshold is chosen by testing several values, and selecting the one that produces the split with the highest information gain.
- For each branch that was created, check which classes appear in the subset of instances in the branch, and calculate its information entropy:

$$entropy = -\sum_{l \text{ is in branch}} p(c_l) \times log_2\, p(c_l)$$

The probabilities in this equation are the class frequencies in the branch:

$$p(c_l) = \frac{number\ of\ instances\ in\ the\ branch\ with\ class\ l}{total\ number\ of\ instances\ in\ the\ branch}$$

- Calculate the weighted average of the entropies of the n branches that were created. The weights are proportional to the number of instances in each branch:

$$entropy_{after} = \sum_{i=1}^{n} w_i \times entropy_i$$

$$where \quad w_i = \frac{number\ of\ instances\ in\ branch\ i}{total\ number\ of\ instances\ in\ the\ n\ branches}$$

- Calculate the entropy in the node before branching, and use this value to determine the information gain when the attribute is used to split the instances:

$$informationGain = entropy_{before} - entropy_{after}$$

Once the information gain of all the attributes has been calculated, the one with the highest gain, i.e., the attribute that produces the split with the lowest entropy, is chosen to divide the instances.

This creates a new set of branches in the tree, and the corresponding nodes are expanded with the same algorithm. The recursion stops when all the instances in the branch are of the same class (in which case a leaf node is created for that class), or when none of the potential splits results in an information gain.

It is possible that the described tree induction algorithm will generate a model that overfits the training data, i.e., an excessively big decision tree that contains the noise in that dataset. To overcome this problem, the tree needs to be pruned. With pre-pruning, the growth of the tree is stopped before it can classify all the training instances correctly; several criteria may be used to decide when to stop splitting the data – for example, the algorithm might create a leaf node when the number of instances that need to be classified is small, or when the information gain of further splits is negligible. With post-pruning, the tree is pruned after it has been induced. Post-pruning is achieved with subtree replacement or subtree raising: subtree replacement means replacing a subtree with a leaf node, and subtree raising means moving a subtree to a higher point in the decision tree, to replace an existing node. A common strategy to determine if a subtree should be post-pruned is to test the model's accuracy before and after pruning it, and then deciding accordingly. This strategy was devised by Quinlan (1987), and is called reduced error pruning. It employs a very simple algorithm – first, it starts by dividing the available data into a training set and a validation set; then, it creates the tree with the training set, and post-prunes it with the following sequence of steps:

- for each node, use the validation set to evaluate the model's accuracy when the node is pruned;
- remove the worst node (i.e., the one whose removal most improves the validation set accuracy);
- repeat the algorithm until further pruning decreases the accuracy.

The fact that the C4.5 algorithm is able to perform its own attribute selection (because the entropy of the splits of irrelevant attributes will be too high for them to become a part of the tree), and that it is able to find patterns based on inter-attribute relationships (each root-to-leaf branch is one such pattern) are some the reasons why the C4.5 decision tree is one of the best data mining tools currently available. In contrast with models that operate like black boxes, it also presents the advantage that its classifications are explicit: we can easily verify why it classified a test instance a certain way, by looking at the corresponding branch in the tree.

Another influential data mining algorithm is the CART (Breiman *et al.*, 1984), short for classification and regression tree. It combines a decision tree inducer for nominal classes, similar to the C4.5 algorithm, and a mechanism to induce regression trees for numeric classes. It differs from the C4.5 inducer in that it only outputs binary trees, and in that, among other differences, it chooses the attributes to split the data according to the Gini impurity measure:

$$gini = 1 - \sum_{l\ is\ in\ branch} \left(p(c_l)\right)^2$$

In each iteration, the classification tree inducer will pick the split with the smallest Gini index. The regression tree inducer, on the other hand, selects the attributes that will grow the tree by looking at the error generated by the corresponding splits, which it calculates with the sum of squares method. Leaf nodes in the regression tree contain the average class value of the training instances in the branch, rather than class labels.

Another model with a similar internal structure is the best-first decision tree (Shi, 2007), which is always binary. Rather than using the depth-first expansion strategy of the C4.5 algorithm, the best-first inducer can expand any node while growing the tree. More specifically, in each iteration, it will select the best possible split anywhere in the tree; to compare the potential splits, it can either calculate the entropy (like the C4.5 model) or the Gini index (like the CART model).

So far, we have only referenced models whose internal structure resembles the sample tree depicted in Figure 52, where the branches represent conjunctions of conditions, and the leaves represent classifications. In an alternating decision tree (Freund & Mason, 1999) these parts have a different meaning. This model is a binary classifier composed of two types of nodes: the decision nodes, which specify conditions, and the prediction nodes, which contain values that are used in the classification task. Figure 53 shows an example of one of these trees, with three decision nodes and three attributes (a_1 and a_3 are nominal, and a_2 is numeric). Given an out-of-sample instance of class c_1 or c_2, the model performs its classification by adding up the values of the prediction nodes in all the branches whose conditions are satisfied by the instance's attributes; if the total sum is positive, the model predicts it belongs to class c_1, otherwise it outputs class c_2. Notice that, unlike in previously described decision trees, the instance will follow more than one path in the alternating tree. The tree is grown using a boosting algorithm. In each iteration, a new condition node and the corresponding pair of prediction nodes are added to the tree; the condition node can be appended to any of the prediction nodes in the tree, and is chosen according to the weighted error.

Besides decision trees, there is one other group of data mining models that exhibit a tree-like internal structure. They differ from the former in that they have regression or classification models as leaves. For example, a naïve Bayes tree (Kohavi, 1996) is a classifier which has a naïve Bayes model in each leaf. This tree is induced using a variation of the C4.5 recursive partitioning

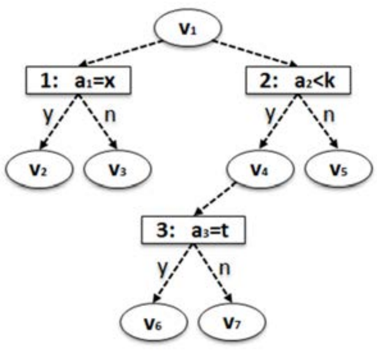


Figure 53. Graphical representation of an alternating decision tree.

algorithm, where the splits are chosen according to the weighted sum of the utility of the corresponding nodes (the utility of each node is given by the 5-fold cross-validation accuracy of a naïve Bayes model in that node, and the weight is proportional to the number of training instances in it). When presented with an out-of-sample instance, the tree will classify it using the naïve Bayes model in the leaf of the branch that matches the attribute values of that instance. Figure 54 shows the basic structure of one of these trees.

There are several other models with a similar structure to the naïve Bayes tree. These include the logistic model tree (Landwehr *et al.*, 2005), a classifier with logistic regression models in its leaves, and the M5 model tree (Quinlan, 1992), a regression tree with linear regression models as leaves. This last one might be particularly useful because, just like the CART model, it applies to problems with numeric classes.

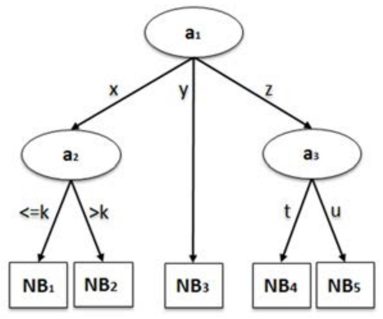


Figure 54. Graphical representation of a naïve Bayes model tree.

4.1.4 Rule Inducers

Rule inducers in general use a covering strategy to learn from the training data (as opposed to the top-down, divide-and-conquer method of decision tree inducers). Simply put, these models take one class at a time, and attempt to create a rule that covers as many instances of that class as possible, by iteratively adding conditions to it. While a split in a decision tree applies to all the classes, a new condition in a rule applies only to the class being targeted. The method for picking the best attribute in each iteration is also different: tree inducers perform attribute selection based on criteria such as the information gain or the Gini impurity measure, while rule learners will simply choose the attribute that maximizes the accuracy for the desired class.

The RIPPER rule learner (Cohen, 1995) is an example of a model that employs a covering strategy. RIPPER (short for "repeated incremental pruning to produce error reduction") is a propositional rule inducer that outputs a rule set; each rule in the set consists of a conjunction of attribute conditions (the antecedents) and the corresponding class value (the consequent). Rule creation is accomplished by greedily adding antecedents to a rule until it becomes 100% accurate. In each iteration, the condition to be added is selected by testing every possible value for each attribute, and picking the condition with the highest information gain, as given by:

$$informationGain = p \times \left(log\left(\frac{p}{t}\right) - log\left(\frac{P}{T}\right)\right)$$

where t is the total number of instances covered by the new rule, p is the number of instances that are correctly classified by it, T is the total number of instances covered by the rule before adding the condition, and P is the amount of those that were classified correctly. The RIPPER rule learner prevents overfitting of the training data by performing incremental reduced error pruning; as previously outlined, this involves using one subset of data to create the rule, and another to test its accuracy when some conditions are pruned. The strategy is called incremental because each rule is pruned immediately after it has been created.

The ripple-down rule learner (Gaines & Compton, 1995) uses a different covering method. It starts by inducing a default rule that maps to one of the classes in the training data. Then, it employs incremental reduced error pruning to iteratively create exceptions to that rule. These exceptions are rules that map to classes other than the default.

In addition to the covering method, there are a few other strategies for generating rules. The M5 decision list (Holmes *et al.*, 1999), for example, uses the following algorithm: in each iteration, an M5 model tree is built, and the best leaf in the tree is turned into a rule; the training instances covered by this rule are removed from the dataset, and the process is repeated. The recursion stops when all the training instances are covered by the rules in the list. As we mentioned in the previous section, M5 model trees have linear models in their leaves; hence, an M5 decision list consists of an ordered set of rules whose antecedents are attribute conditions and whose consequents are linear models, which means this model is intended for the prediction of numeric class values.

The PART decision list (Frank & Witten, 1998) is similar to the M5 decision list, in that it combines the two main methods for rule induction: the covering strategy, and rule extraction from decision trees. In each step of the learning process, it creates a C4.5 decision tree and converts its best leaf into a rule. The criterion for picking the best leaf is based on the total number of instances covered: the more, the better. The instances covered by the new rule are eliminated from the training data, and the process is repeated until all the instances are covered.

Another way to generate rules is to create a table where each column is an attribute, and each row is a training instance. That is more or less what the decision table majority classifier (Kohavi, 1995b) does. This model consists of a decision table, coupled with a default rule that maps to the most frequent class in the training set. In order to create the table, the inducer starts by discretizing the numeric attributes in the training data; next, it uses a generic attribute selection algorithm that performs cross-validation to choose the best subset of training attributes (i.e., it selects the attributes that best predict the class, with the least redundancy between them). These attributes become the columns in the table, which is then populated with the training data – each row is a simple rule containing the discretized attribute values of a training instance, and the corresponding class. In order to classify an out-of-sample instance, the algorithm searches for all the rows that match the attributes of that instance, and the class is chosen by majority vote. If no match is found, the default rule makes the classification, meaning the most frequent class in the training set will be outputted.

Finally, we present one last method for extracting rules from training data. The fuzzy lattice reasoning classifier (Kaburlasos *et al.*, 2007) employs hyperbox-based rule induction to perform classification. More specifically, it partitions the feature space using hyperboxes, assigns a class to each hyperbox, and then uses these partitions as rules to perform classification: if an out-of-sample instance i is inside a hyperbox labelled with class c, then the class of i is c. The classifier learns by progressively increasing the size of the hyperboxes: for each training instance, it calculates the diagonal size increase that the hyperboxes of that class require to reach the instance in the feature space, up to a maximum user-defined threshold, and then enlarges the hyperbox that requires the smallest change. Similarly, the non-nested generalised exemplars model (Martin, 1995) is a type of a nearest neighbour algorithm that also generalizes hyperrectangles from the training instances, which it converts to if-then rules to classify out-of-sample data.

4.1.5 Perceptron Models

The perceptron (Rosenblatt, 1958) is an algorithm that aims to find a hyperplane in the feature space that can separate training instances belonging to a binary class (with possible values $+1$ and -1). The equation for the hyperplane is:

$$w_0 + w_1 \times a_1 + w_2 \times a_2 + \cdots + w_n \times a_n = 0$$

where $w_0, \ldots, w_n$ are weights and $a_1, \ldots, a_n$ are the attributes. Figure 55 presents a graphical representation of a perceptron with these parameters. This figure shows that the perceptron is the simplest type of artificial neural network, with the input layer connecting directly to the output layer; for binary classification, f is a threshold activation function:

$$f(a_1, \ldots, a_n) = \begin{cases} +1 \quad if \quad w_0 + \sum_{i=1}^{n} (w_i \times a_i) > 0 \\ -1 \quad otherwise \end{cases}$$

This classification function is equivalent to putting the instance in the feature space, and checking in which side of the hyperplane it ends up – on one side, it gets classified with class $+1$, and on the other with class -1.

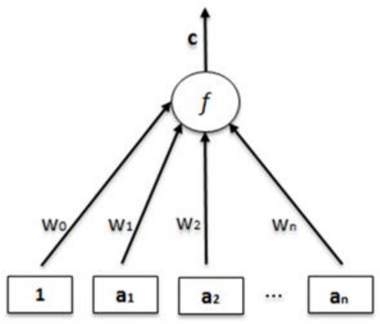


Figure 55. Graphical representation of a perceptron.

To calculate the weights of the hyperplane that separates the two classes, the learning algorithm starts by setting them all to zero, or to random values. Next, the perceptron will classify the training instances one by one. During this process, if an instance of class $+1$ is misclassified, the values of its attributes are added to the corresponding weights; if, on the other hand, an instance of class -1 is misclassified, the attribute values are subtracted from the weights. This procedure moves the hyperplane, so that the instance ends up on the correct side of the separation, or at least closer to it. These steps are repeated until the perceptron is able to classify all the training instances successfully, or until a predefined maximum number of iterations is reached. The iteration limit guarantees that the learning algorithm will always terminate, even if the classes are not linearly separable. Note that if the algorithm is started with random weights, multiple runs may generate different separating hyperplanes for the same training data, which will affect the accuracy of the model when classifying out-of-sample instances.

When the two classes are not linearly separable, the perceptron learning rule can create nonlinear decision boundaries by employing the kernel trick (Aizerman *et al.*, 1964). This strategy implies mapping the training instances to a higher-dimensional space, where linear classification is to occur. The kernel trick introduces some changes to the learning algorithm. As previously described, the perceptron will classify an out-of-sample instance with attributes $a_1, \ldots, a_n$ by checking if the following expression is greater than zero:

$$\sum_{i=0}^{n}(w_i \times a_i)$$

where a_0 is 1 and the weights $w_0, \dots, w_n$ are calculated during training. Considering how the calculation of the weights is accomplished (by adding or subtracting the attributes of the misclassified instances) we can rewrite the expression as:

$$\sum_{i=0}^{n}\left(\sum_{j\ is\ misclassified}\left(y^{(j)} \times a_i^{(j)} \times a_i\right)\right)$$

where $y^{(j)}$ is $+1$ or -1, depending on the class of the jth misclassified training instance, and $a_i^{(j)}$ is the ith attribute value of that instance. The classification of a test instance using this new expression requires iterating through all the misclassified training instances, rather than simply using the pre-calculated vector of weights. Nevertheless, this transformation is advantageous because it allows us to use the dot product in the calculation. Swapping the summation signs, we get:

$$\sum_{j\ is\ misclassified}\left(y^{(j)} \times \sum_{i=0}^{n}\left(a_i^{(j)} \times a_i\right)\right)$$

In mathematics, the dot product of vectors a and b is defined as:

$$a \cdot b = \sum_{i=0}^{n}(a_i \times b_i)$$

Thus, the previous expression may be simplified to:

$$\sum_{j\ is\ misclassified}\left(y^{(j)} \times \left(a^{(j)} \cdot a\right)\right)$$

The kernel trick consists in replacing the dot product between the two vectors with a kernel function k:

$$\sum_{j\ is\ misclassified}\left(y^{(j)} \times k\left(a^{(j)}, a\right)\right)$$

Now, if the two classes are not linearly separable, we can use a kernel function to map the instances to a higher-dimensional space, where that separation might be possible. Linear classification in the new space will correspond to nonlinear classification in the original space. Several kernel functions are available for this purpose, among which polynomial, sigmoid and Gaussian radial basis kernels.

Various data mining models employ the kernel trick to perform nonlinear classification. The voted perceptron (Freund & Schapire, 1999) is one of them. This classifier aims to improve the accuracy of the perceptron, by making some changes to its learning method. During training, the model saves all the different weight vectors calculated by the perceptron's supervised learning algorithm. For each vector, it records the number of iterations it "survived" before being replaced with a new vector. As previously outlined, this replacement occurs whenever there is a misclassification. When the time comes to predict the class of an out-of-sample instance, each of these weight vectors is used to classify it, and the final classification is decided by weighted voting, with the weight of each vote being based on the survival time of the corresponding vector.

The multilayer perceptron (Rumelhart *et al.*, 1986), a type of feedforward artificial neural network, is also an improvement to the perceptron. It has at least three layers: the input layer, the output layer, and one or more in-between hidden layers. The information flows in just one direction, from the input nodes to the hidden nodes, and finally to the output nodes. These nodes are commonly referred to as neurons. Figure 56 depicts a multilayer perceptron with three layers; it has two input attributes, four hidden neurons (with nonlinear activation functions t), and two output neurons (with threshold activation functions f).

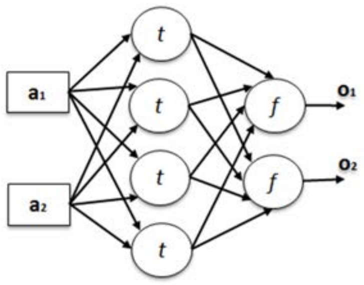


Figure 56. Graphical representation of a multilayer perceptron.

A multilayer perceptron is capable of performing nonlinear classification because the outputs of its hidden neurons are calculated by nonlinear activation functions. Several functions can be utilized, the only requirement being that they are normalisable and differentiable. The sigmoid function, which returns a value between 0 and 1, is the most frequent choice. The output o of an artificial neuron with a sigmoid activation function is:

$$o = \frac{1}{1 + e^{-(w_0 + w_1 \times i_1 + w_2 \times i_2 + \cdots + w_n \times i_n)}}$$

where $i_1, \ldots, i_n$ are the neuron's input values and $w_0, \ldots, w_n$ are the weights of the corresponding connections, or synapses. These weights, initially set to random values, are calculated during training using a supervised learning method, like the backpropagation algorithm. This algorithm consists of the following sequence of steps:

- Feed a training instance to the artificial neural network.
- Calculate the error in each output neuron. If neuron i outputs o_i, and the expected value is y_i, then the neuron's error e_i is:

  $$e_i = o_i \times (1 - o_i) \times (y_i - o_i)$$

- Calculate the error in each hidden neuron. For hidden neuron q, this is given by:

  $$e_q = o_q \times (1 - o_q) \times \sum_{i \; is \; forward \; connected} (w_i \times e_i)$$

  Notice the error in the hidden neuron depends on the errors in the neurons it is forward-connected to. Hence, the errors are being propagated backwards from the output layer.

- Optimize the weights of the synapses. The new weight for synapse s that connects neuron k to neuron t is:

  $$w_s = w_s + \eta \times o_k \times e_t$$

  where η is the user-defined learning rate, which speeds up or slows down the learning.

- Repeat until all the training instances are classified correctly, or another stopping criterion is satisfied.

As is, this algorithm applies only to multilayer perceptrons where the neurons have sigmoid activation functions; other functions would require different formulas to calculate the errors in the neurons.

The radial basis function network (Moody & Darken, 1989) is another type of feedforward artificial neural network. It consists of an input layer, a hidden layer and an output layer. The activation of the hidden neurons is calculated with radial basis functions. From a practical point of view, each hidden neuron in this network can be visualised as a point in the feature space. When fed with an out-of-sample instance to classify, the output of each hidden neuron will depend on its distance to the instance: the closer they are in the feature space, the stronger the activation.

4.1.6 Miscellaneous Models

The naïve Bayes classifier (John & Langley, 1995) is one of the best known tools in data mining. This statistical model is based on Bayes' theorem of conditional probability – it calculates the probability of an instance with attribute values $a_1, \dots, a_n$ belonging to nominal class c with the formula:

$$P(c|a_1, \dots, a_n) = \frac{P(c) \times P(a_1, \dots, a_n|c)}{P(a_1, \dots, a_n)}$$

The model is considered "naïve" due to its assumption that the attributes are independent from each other, given the class. This assumption is incorrect for most real life scenarios. Still, it makes the classification task much simpler, by allowing the simplification of the numerator:

$$P(c) \times P(a_1, \dots, a_n|c) = P(c) \times P(a_1|c) \times P(a_2|c) \times \dots \times P(a_n|c)$$

Since the denominator does not depend on the class, it will remain the same when we calculate the probability of the test instance belonging to each class. Thus, we can replace it with a constant that ensures that these probabilities add up to 1. Considering all these simplifications, if n is the number of attributes and t is the number of classes, the naïve Bayes model will calculate the probably of a test instance belonging to class c_k with:

$$P(c_k|a_1, \dots, a_n) = \frac{P(c_k) \times \prod_{i=1}^{n} P(a_i|c_k)}{\sum_{j=1}^{t}\left(P(c_j) \times \prod_{i=1}^{n} P(a_i|c_j)\right)}$$

The classes' prior probabilities $P(c_1), \dots, P(c_t)$ are calculated by counting the number of times they occur in the training set, and computing their relative frequencies. If a_i is a nominal attribute, then $P(a_i|c_k)$, i.e., the probability that a given label of a_i occurs with class c_k, is calculated the same way. If a_i is a numeric attribute, it may also be treated the same way, if we discretize it first. Otherwise, it is assumed that the values of a_i follow a normal distribution, and the training data is used to calculate the mean and the standard deviation of those values for class c_k. The probability $P(a_i|c_k)$ can then be determined by looking at the probability density function of the corresponding normal distribution. If a numeric attribute does not seem to follow a Gaussian distribution, other distributions may be utilized instead.

Once all the prior and conditional probabilities have been computed from the training set, the naïve Bayes model is ready to classify out-of-sample instances. Its strategy is straightforward: for an instance with attributes $a_1, \dots, a_n$, it uses the formula to compute the probability that it belongs to each of the possible classes, i.e., $P(c_1|a_1, \dots, a_n), \dots, P(c_t|a_1, \dots, a_n)$, and picks the class with the highest probability.

Another renowned data mining model is the support vector machine (Boser *et al.*, 1992). Given a binary classification problem with classes c_1 and c_2, the training of a support vector machine entails calculating the maximum margin hyperplane that can separate the two classes in the feature space. The maximum margin hyperplane is the hyperplane that provides the greatest separation between the classes, as exemplified in Figure 57. The classification of an out-of-sample instance can be envisioned as putting it in the feature space, and checking in which side of the hyperplane it ends up. In the example in Figure 57, the represented support vector machine would classify the out-of-sample instance (represented in grey) as belonging to the *WHITE* class.

The training instances that are closest to the maximum margin hyperplane are called support vectors (those in the dashed lines in Figure 57). These are the only instances that are needed to define the maximum margin hyperplane. The classification of an out-of-sample instance with attribute values in vector a is accomplished by calculating if the following expression is greater than or less than zero:

$$b + \sum_{j \text{ is support vector}} \left(\alpha_j \times y^{(j)} \times k\left(a^{(j)}, a\right) \right)$$

where $y^{(j)}$ is $+1$ if instance j belongs to class c_1 or -1 if it belongs to class c_2, $a^{(j)}$ is the vector with the attribute values of j, and b and α are parameters calculated by performing constrained quadratic optimization on the training set. Notice the support vector machine employs the kernel trick to do the classifications. To define linear class boundaries in the original space, the following kernel is utilized:

$$k\left(a^{(j)}, a\right) = a^{(j)} \cdot a$$

Nonlinear classification is achieved by using a kernel function that maps the instances to a higher-dimensional space, where the maximum margin hyperplane is to be found. The polynomial kernel is one possibility:

$$k\left(a^{(j)}, a\right) = (a^{(j)} \cdot a + 1)^d$$

Support vector machines address multiclass problems by first converting them to binary problems. There are several ways to do this; the most common strategy is to create a support vector machine for each pair of classes, and then classifying out-of-sample instances by majority voting.

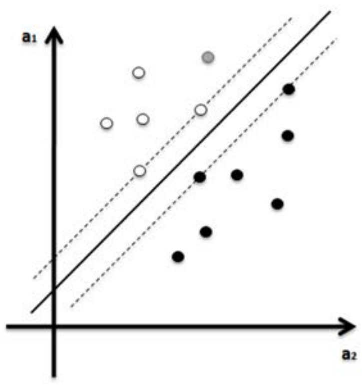


Figure 57. Graphical representation of the maximum margin hyperplane for a set of instances with two attributes.

Naïve Bayes classifiers and support vector machines are some of the most important tools in data mining. So much so that they were picked as two of the top 10 algorithms (Wu *et al.*, 2007), along with others that were previously referenced, like the C4.5 decision tree, the k-nearest neighbour and the CART. Some obscure models have not achieved the same status, but that does not mean they cannot be useful. The voting feature intervals classifier (Demiröz & Güvenir, 1997), for example, is one of the simplest ways to parse data. It starts by splitting each attribute in intervals: numeric attributes are discretized; for nominal attributes, a single point interval is created for each possible label. Next, it counts the number of times each class occurs with each interval in the training instances. Classification of a test instance is achieved by determining the intervals to which its attribute values belong, adding the intervals' frequencies for each class, and selecting the class with the biggest sum. This strategy is not very sophisticated, but may still prove useful for simple data mining problems.

The distance to average classifier, which we authored, is also extremely simple. It is somewhat similar to the nearest neighbour model, only it uses an eager strategy, rather than a lazy strategy. This means it does the bulk of the work during training, so that the classification task can be performed faster. The training phase is itself relatively fast, because all the model does is calculate the coordinates of the mean point that best represents the training instances of each class in the feature space. This strategy is easily explained with an example. Suppose we want to predict if the price of a stock will increase or decrease tomorrow. We have historical price data for six days, which we convert into six training instances, with three numeric attributes each: the RSI and the Williams %R technical indicators, and the percentage price change for the day. The class value of each instance is set to *UP* or *DOWN*, depending on whether the price of the stock increased or decreased the following day. The placement of the six instances in the feature space is depicted in Figure 58; instances of class *UP* are represented in white, while instances of class *DOWN* are represented in black. Notice the feature space is three-dimensional, because the training instances

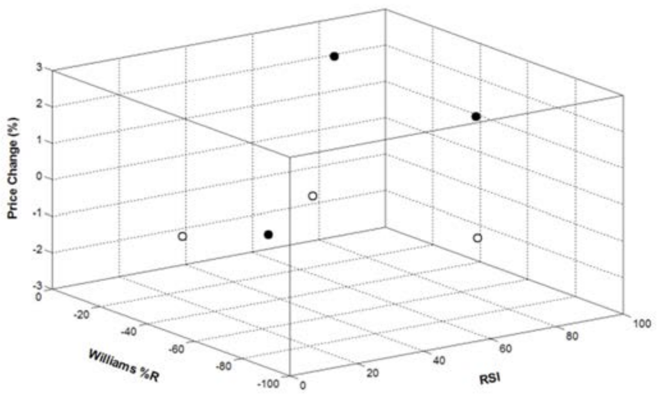


Figure 58. Training instances in the feature space.

are composed of three attributes. The construction of the distance to average model is accomplished by calculating the mean point for each class in that space. The coordinates of the mean point of a class are given by the average attribute values of the training instances belonging to that class. Figure 59 shows the mean points for the *UP* and the *DOWN* classes, represented with stars. Once the central points of all the classes have been calculated, the model is ready to make predictions. Given a new instance, its classification is performed by putting it in the feature space, and calculating the Euclidean distance between it and the mean point of each class; the class whose mean point is closest is chosen as the class prediction for the instance.

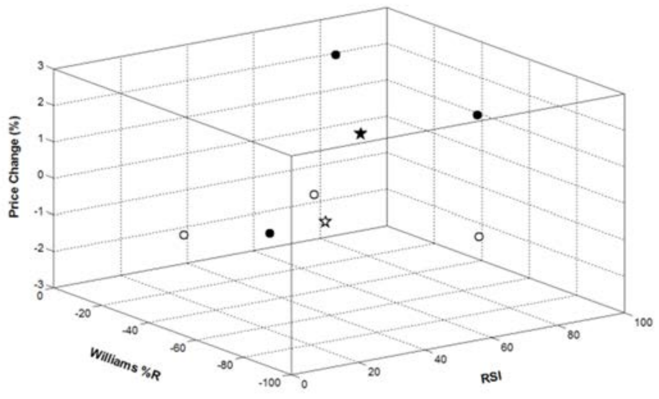


Figure 59. Mean points of the classes in the feature space.

Figure 60 demonstrates the classification of an out-of-sample instance (shown in grey) using the distance to average model. Since the instance is nearest to the central point of the *UP* class, the model will predict it belongs to that class, which in practical terms means it will predict a stock price increase for the following day.

The distance to average model normalizes numeric attribute values before calculating the distances, so that the attributes' contribution to the distance does not depend on their scale. It also supports nominal attributes, by using the training instances to compute, for each class, the frequency of each label. When calculating the Euclidean distance between a test instance and the central point of a class, the contribution of the nominal attribute will be 1 minus the frequency with which the label occurred with that class in the training data; thus, the higher the frequency, the smaller the distance. The training and the classification methods of the distance to average model are described in Algorithm 10. Please notice this model was presented just for demonstration purposes, as there are much more advanced algorithms for calculating prototypes from the training instances (Chang *et al.*, 2006).

Since the distance to average model gives the exact same importance to all the attributes when classifying new data, the presence of redundant or irrelevant attributes in the training instances will degrade its performance. Thus, the accuracy of the model will mostly depend on the user's ability to

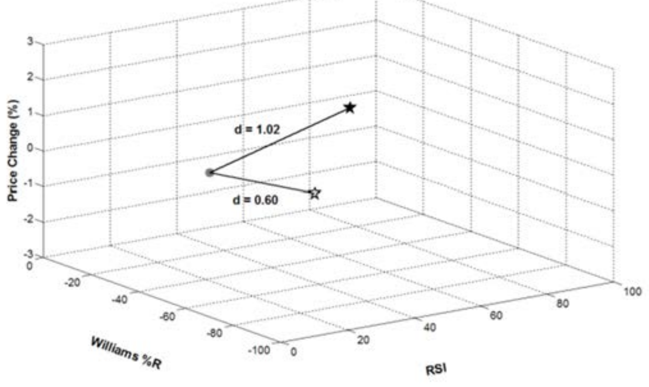


Figure 60. Classification of an out-of-sample instance according to the normalized Euclidean distance to the mean points of the classes.

properly select the training attributes. Several other models that were discussed in this chapter also suffer from this inability to give more weight to the attributes that best predict the class. Regardless, even the most sophisticated data mining algorithms will perform badly, if the training instances do not contain enough useful information. For this reason, in addition to being able to pick the right model for the job, the user must also be proficient at pre-processing the raw data (to fix incomplete, inconsistent or noisy information), and at selecting the best training attributes. The preparation of the data is, in fact, the most important task when doing data mining. In the next section, we will present the list of attributes that we chose for transforming the raw financial data into training instances. These instances will be used later to train the models that make up the agents' ensembles.

```
Method BuildModel
Input: training instances
    Calculate the prior probability of each class
    For each attribute:
        If the attribute is numeric:
            calculate the average for each class
            normalize the average values so that they lie between 0 and 1
        Else if the attribute is nominal:
            calculate the frequency of each label for each class

Method ClassifyInstance
Input: test instance
Output: class prediction
    For each class:
        For each attribute:
            If the attribute is numeric:
                normalize the value of the attribute
                calculate the difference between the normalized value and the class's mean value for that attribute
            Else if the attribute is nominal:
                calculate the difference between 1 and the frequency with which the label occurred with the class
            Add the square of the calculated difference to the total distance between the instance and the mean point of the class
    If there is a draw regarding the class whose central point is closest to the instance:
        return the one with greater prior probability
    Else:
        return the class whose central point is closest to the instance
```

Algorithm 10. Pseudocode for the distance to average classifier.

4.2 Data Mining Attributes

Successful data mining, i.e., the discovery of useful hidden patterns in raw empirical data, is far from an easy feat. It involves two steps:

- selecting the attributes with which the empirical data should be converted into training instances;
- choosing the algorithm that will perform the data mining, along with its training parameters.

In the previous section, we briefly described the strategies employed by many different types of algorithms to model the training data. Understanding how these algorithms work (i.e., how they learn and generalize from the training instances), and how changing their parameters impacts the performance of the resulting models, is an important part of the data mining process. Nonetheless, the real skill in this process is shown when transforming the raw data into the training datasets that the algorithms will attempt to model. This implies selecting the nominal or numeric attributes that best describe the raw data, and filtering out the irrelevant or redundant information. The importance of this step cannot be stressed enough. If neither the attributes themselves, nor the relationships between them, can offer any insight into the instances' class value, the predictions of the model will be worthless. A data mining algorithm will not be able to find usable patterns in the data if there are none to be found. The relationships it does find, when processing irrelevant attributes, will bear no connection to the underlying data generating process, which means the model will be an awful predictor of out-of-sample data. Simply put, if the attribute selection is not done properly, the "garbage in, garbage out" mantra will apply.

For financial forecasting, there are several attributes that we believe might be useful, like those that describe the fundamentals of the financial instruments. For example, if the object of study is a company's stock price, we could populate the training instances with ratios such as the P/E (price to earnings) or the EPS (earnings per share), the company's debt, revenue, projected growth, and so forth. For a currency pair, we could consider attributes like the differential between the interest rates set by the central banks overseeing the two currencies, the external debt and gross domestic product of the corresponding countries or zones, the growth in their money supplies, etc. Clearly, there are many different bits of information that would suit this problem well. However, when we devised the iQuant software, our plan was to make it as generic as possible, so that it could be applied effortlessly to different financial instruments in different markets. For this reason, we decided to disregard any instrument-specific attributes based on fundamentals, and use only the type of information that is available for all financial products: historical prices. According to the proponents of technical analysis, this is actually an ideal strategy. Technical analysts believe that an instrument's price always fully reflects its fundamentals, so historical prices should be all that is needed to forecast future prices. While we do not necessarily agree with this point of view, we decided to follow this route, and define only time and price-based attributes to train the models. This might limit their accuracy, but makes it a lot easier to compare the usefulness of the proposed agent architecture across different markets. By only allowing generic attributes, all that is necessary to create an intelligent agent with the iQuant software is some historical price data, regardless of the type of instrument that it will trade.

Since the iQuant software uses the Weka API to train the data mining models and to get their predictions, the files with the instances' information need to be in the attribute-relation format, or ARFF. Figure 61 shows the content of a sample ARFF file, with some training instances. More specifically, this listing contains information for 6 instances, each corresponding to a trading period of 6 hours. The instances are comprised of 8 attributes (one nominal and 7 numeric), plus the class label, which describes the price movement in the trading period that followed (*UP* if the price increased, *DOWN* if it decreased). This example shows just a small subset of the attributes offered by the iQuant software to train the agents' ensembles; the software is able to automatically extract

```
@RELATION usdjpy_6h
@ATTRIBUTE      CLOSINGPRICE     NUMERIC
@ATTRIBUTE      HOUR             NUMERIC
@ATTRIBUTE      DAYOFWEEK        { Monday, Tuesday, Wednesday, Thursday, Friday, Sunday }
@ATTRIBUTE      WR30             NUMERIC
@ATTRIBUTE      RSI20            NUMERIC
@ATTRIBUTE      MA10             NUMERIC
@ATTRIBUTE      LAG1             NUMERIC
@ATTRIBUTE      PRICECHANGE      NUMERIC
@ATTRIBUTE      NOMCLASS         { up, down }

@DATA
116.76,  18,  Tuesday,     -73.0,  42.0,  -0.0331,  -0.2311,   0.163,    down
116.65,  0,   Wednesday,   -77.0,  45.0,  -0.0502,   0.163,   -0.0942,   up
116.74,  6,   Wednesday,   -74.0,  54.0,  -0.0263,  -0.0942,   0.0772,   down
115.88,  12,  Wednesday,   -90.0,  44.0,  -0.0794,   0.0772,  -0.7707,   up
116.28,  18,  Wednesday,   -77.0,  46.0,  -0.0218,  -0.7707,   0.3538,   down
116.17,  0,   Thursday,    -81.0,  49.0,  -0.0587,   0.3538,  -0.0946,   down
```

Figure 61. Sample ARFF file.

the values for these and many other attributes from historical price data. In the next sections, we will describe all the different generic attributes available to the agents, and will also present the reasons why we think each of them might be useful for predicting future data.

4.2.1 The Class Feature

The nominal class feature is the attribute that the classification models will try to predict. It describes the direction of the instrument's price throughout the next trading period, and has two possible values: "the price increased in the next trading period" (*UP*) or "the price decreased in the next trading period" (*DOWN*). Concretely, the nominal class of the training instance corresponding to trading period i is:

$$class_i = \begin{cases} UP \ if \ \ (close_{i+1} - open_{i+1}) \geq 0 \\ DOWN \ \ otherwise \end{cases}$$

where $open_{i+1}$ and $close_{i+1}$ are the instrument's open and close prices in the following period. When a model classifies an instance of an unknown class, it is actually making a prediction regarding what will happen to the price in the future: an *UP* prediction means it expects the opening price to be lower than or equal to the closing price in the next period, while a *DOWN* prediction means it expects it to be higher.

In addition to classification models, we also wanted the agents to use regression models. Hence, we defined a numeric class with which they could be trained. The value of this attribute is equal to the percentage price change in the next trading period, from open to close. For the instance representing period i, the numeric class value is:

$$class_i = \frac{close_{i+1} - open_{i+1}}{open_{i+1}} \times 100$$

When predicting the class value of an out-of-sample instance, the regression model will output the percentage price change it expects for the subsequent period. Since the agents' prediction modules require price direction forecasts, they will convert these numeric predictions into nominal labels: if the regression model predicts a positive change, its nominal prediction is *UP*, otherwise it is *DOWN*.

4.2.2 Time-Based Attributes

We defined a few time-based attributes, so that the agents could find relationships between temporal information (hour of the day, day of the week, month, etc.) and the prices of financial instruments. The "hour of the day" attribute, which can either be nominal or numeric, is the starting hour for the period represented in the instance. Its value will depend on the agent's investment time frame. For example, for Forex agents trading with a 6-hour time frame, the possible values are: 0, 6, 12 and 18. For stock agents trading with a 24-hour time frame this attribute will be redundant, and thus should not be used: since all the training instances will have the same value for the starting hour, the data mining models cannot utilize that information to distinguish the classes.

The reason why we allowed the hour to be represented as either a nominal or a numeric attribute was that not all data mining algorithms support both types. By making them both available, we ensured that all the models would be able to process the "hour of the day" information, regardless of their limitations. We should point out that, for models that support both types, the decision to represent the hour as a nominal or a numeric attribute is not irrelevant. Even if they do carry the same information, some models will treat nominal and numeric attributes differently during training, so picking one type over the other may affect their performance.

Several studies have reported the existence of weekday seasonality in stock prices (Harris, 1986; Pettengill, 2003). It is unlikely that this pattern could be employed, by itself, to develop a profitable trading strategy, even more so because some other studies have denied the existence of this effect in mature markets (Prokop, 2010). Nevertheless, we decided to let the agents use the "day of the week" attribute. Even if it cannot directly predict the class, it is possible that there are relationships between this and other attributes that might correlate with the class. This attribute was also made available in both nominal and numeric form. Possible values for the nominal attribute are: Sunday, Monday, Tuesday, Wednesday, Thursday and Friday. For the numeric attribute, the equivalent values are: 0, 1, 2, 3, 4 and 5.

The last time-based attributes that we defined were the numeric "day of the month" and the "month of the year". It is not much of a stretch to imagine that the instruments' prices could be affected by these variables. In fact, there is an old Wall Street adage that goes: "sell in May and go away". This is known as the Halloween indicator; it implies that stock market returns are worse between May and October, compared to the period between November and April. Some studies have confirmed the statistical significance of this seasonality in stock prices (Jacobsen & Visaltanachoti, 2009), hence it makes sense that we would let the agents use this temporal information to find patterns.

4.2.3 Price-Based Attributes

The idea that historical prices can predict future prices is controversial. Those who believe in efficient markets call it a fool's errand, while technical analysts defend the opposite and swear by their methods. There is evidence supporting both views in academic literature. Consider the theory of mean reversion, which states that prices or returns have a tendency to move back towards their historical average. We can find studies that support this theory with empirical evidence (Balvers *et al.*, 2000; Chortareas *et al.*, 2002), as well as studies rebutting it (Kim *et al.*, 1991; Miller *et al.*, 1994). We decided to side with the technical analysts on this issue, and assume that past price patterns will indeed reappear in the future, be it due to economic cycles, traders' behavioural predictability, self-fulfilling prophecies (i.e., when many traders believe a price pattern will repeat itself, they will act on it, and cause it to happen themselves), or some other unknown reason. Thus, we added the price direction to the list of usable attributes. This nominal attribute represents the direction of the price in the instance's trading period. It has two possible values: "the price increased in the trading period" (*UP*) or "the price decreased in the trading period" (*DOWN*). More concretely, the label for the instance that represents trading period i is:

$$direction_i = \begin{cases} UP\ if\ \ (close_i - open_i) \geq 0 \\ DOWN\ \ otherwise \end{cases}$$

where $open_i$ and $close_i$ are the opening and closing prices in that period. Note that the direction label of an instance is equal to the class label of the previous instance.

The numeric version of this attribute is the percentage price change (i.e., the return) in the instance's period. It is also calculated using the opening and closing prices:

$$change_i = \frac{close_i - open_i}{open_i} \times 100$$

This attribute differences the price data; this is an important pre-processing transformation, because it removes the trend from the price series, and makes it easier to model (Franses, 1998). The instrument's closing price in each trading period (i.e., the original data, prior to differencing) was also made available to the agents. For stocks, we use the closing price adjusted for dividends and splits (because a split completely changes the price level of the stocks, meaning the original historical prices will no longer be comparable with future prices).

The ARIMA is a classical, time-tested method of modelling time series data. As outlined in Chapter 2, this model starts by differencing the data, and then uses lagged values (i.e., previous values in the series) and moving averages of these lagged values to model the series. We allowed the agents to use this same information by defining the "lagged percentage price change" and the "lagged percentage price change moving average" numeric attributes. The lagged percentage price change is simply the return in a trading period preceding the instance's period:

$$lag(t)_i = change_{i-t}$$

The first lag is the change in the penultimate period, the second lag is the change in the antepenultimate period, and so forth. The moving average attribute is the average return in the last n periods:

$$MA(n)_i = \frac{change_i + \sum_{t=1}^{n-1} lag(t)_i}{n}$$

As we saw in Section 2.1, the moving average attribute is a common choice in financial data mining studies. Going by the articles referenced in that section, technical analysis indicators are also a good choice. We added a couple of these indicators to our list of attributes. The first was the Williams %R (Williams, 1979), an oscillator that compares the closing price in the instance's period with the highest and lowest prices in the last n periods:

$$WR(n)_i = \frac{close_i - high_n}{high_n - low_n} \times 100$$

The value of the Williams %R oscillates between -100 and 0, and is usually interpreted as follows: if it is below -80, the instrument is oversold, hence the price is expected to increase in the future; if it is above -20, the instrument is overbought, and so the price is expected to fall. Obviously, these are just rough guidelines that technical analysts use to interpret the values of this indicator. The data mining models will likely find completely different relationships between the Williams %R and the instrument's price direction.

In order to distinguish rising values from declining values, we defined the signed Williams %R attribute:

$$SWR(n)_i = \begin{cases} |WR(n)_i| & if \quad WR(n)_i \geq WR(n)_{i-1} \\ WR(n)_i & otherwise \end{cases}$$

The second technical analysis indicator in our list of attributes is the relative strength index (Wilder, 1978). This oscillator measures the momentum of directional movement, by comparing upward and downward price movements. It is calculated with the following formula:

$$RSI(n)_i = 100 - 100 \times \frac{1}{1 + RS(n)_i}$$

$where$

$$RS(n)_i = \frac{UPMA(n)_i}{DOWNMA(n)_i}$$

$$UPMA(n)_i = \frac{\sum_j close_j - open_j}{n}, j\ is\ a\ period\ in\ last\ n\ in\ which\ the\ price\ increased$$

$$DOWNMA(n)_i = \frac{\sum_j open_j - close_j}{n}, j\ is\ a\ period\ in\ last\ n\ in\ which\ the\ price\ decreased$$

This indicator oscillates between 0 and 100, with the traditional interpretation being that the instrument is overbought if the indicator is above 70, and oversold if it is below 30. To distinguish increasing values from decreasing values, we also defined the signed relative strength index attribute:

$$SRSI(n)_i = \begin{cases} RSI(n)_i \ \ if \ \ RSI(n)_i \geq RSI(n)_{i-1} \\ \\ -RSI(n)_i \ \ otherwise \end{cases}$$

Finally, the last attribute that we implemented was the rate of change, a simple technical analysis indicator that shows the difference between the closing price in the instance's period, and the closing price n periods before:

$$ROC(n)_i = \frac{close_i - close_{i-n}}{close_{i-n}}$$

The rate of change indicates the continuation of the trend when it remains positive in an uptrend, or negative in a downtrend. If its value changes from negative to positive between periods, this is traditionally seen as a signal to buy the instrument; if the opposite happens, the instrument should be sold.

We have now presented all the different attributes available to the agents. Their usefulness will depend on the financial instrument being traded. It is possible that, for some instruments, none of these attributes will help predict the direction of the price. Historical price data, the agents' only source of information, is very erratic and noisy, so our expectations for their potential accuracy must be kept low. As previously mentioned, adding instrument-specific attributes extracted from other sources, like fundamental data or news feeds, would probably be a good idea, because it is conceivable that there is a relationship between this information and the instruments' prices. We opted not to do it, to keep the iQuant software generic, but it is likely that some of these attributes would make far better predictors than historical prices – that is, at least, the opinion of value investors, whose strategy consist of analysing the fundamentals to find undervalued instruments. In addition to the fundamentals, there are numerous other features that the agents could experiment with, including a few strange ones like the lunar phase (Yuan *et al.*, 2006), the weather (Hirshleifer & Shumway, 2003) and geomagnetic storm information (Robotti & Krivelyova, 2003), all of which have been shown to be related with stock prices. The common idea behind the studies that report these weird relationships is that the investors' mood is affected by those factors, and this influences the way they trade. Regardless, we believe those patterns are probably just fabrications arising from weak data mining. A recent study has reported an equally dubious relation between the moods expressed in Twitter feeds, and the direction of the stock market (Bollen *et al.*, 2010). The authors claimed that a model trained with the "Twitter sentiment" was able to predict the daily up and down changes in the closing values of the Down Jones 30 Index with 86.7% accuracy. This outrageous claim should be a red flag to anyone with a bit of experience in the field, because that accuracy goes well beyond what may reasonably be expected from a financial data mining model. The procedural errors underlying this unbelievable result are easy to pinpoint. First of all, the researchers tested the model with out-of-sample data for the period between December 1st and December 19th, which means the reported accuracy is based on less than 20 predictions. Also, they experimented with several combinations of sentiment attributes and lagged index values, and singled out the combination with the best accuracy; with almost all other combinations, the accuracy they got was less than or equal to the accuracy achieved with just the lagged values. Because of these problems, we can conclude that the results of this study are deceiving, and useless in practice. Its premise was pretty doubtful to begin with, since it is hard to imagine how an amalgamation of tweet messages could ever predict the direction of a stock market index – we can envision the fluctuations of the financial markets affecting the posters' mood, not the other way around. Nevertheless, it has been reported that a multi-million dollar hedge fund is being created, to try to exploit the patterns uncovered by that research (Jordan, 2010); this initiative will probably end very badly. All these studies describing strange and implausible patterns demonstrate just how difficult it is to derive robust and meaningful conclusions from the mining of financial data. Whether consciously or unconsciously, researchers always run the risk of letting their own biases become a part of the process. And that is worrisome, because data mining will easily "confirm" any preconceived idea: with enough tweaking and tampering, data mining algorithms can be made to output results that corroborate the flimsiest of premises. In order to avoid this pitfall, we had to come up with a sound method for selecting the attributes and the models that would comprise the agents' ensembles. After considering all the pros and cons, we decided on an automatic selection strategy, so as to remove the human element from the whole process. This strategy will be described in the next section.

4.3 Unbiased Model Selection

The creation of an accurate data mining model can be very challenging, due to various pitfalls. The most prevalent is data overfitting; a model is said to overfit the training data when it describes the noise in the data, rather than its hidden patterns. For example, if we induced a decision tree for a training set, and the tree ended up with a different leaf for each training instance, it would clearly be overfitting the data. Hence, whereas it would classify all the training instances very accurately, it would not be capable of generalizing from that data, and so its accuracy classifying out-of-sample instances would be very poor. This problem is typical of models that are too complex; that is the reason why the learning algorithms of most decision tree inducers and rule learners include some sort of pruning procedure (which is responsible for simplifying the models by eliminating unnecessary parts). A common strategy for detecting if a model overfits the training data is to evaluate its accuracy with test instances (i.e., unseen data) and then discern if it was able to learn the concept. However, this strategy is only helpful if the test sample is representative of the full population of instances that the model will need to classify in the future.

Another problem that could undermine a data mining effort is the improper selection of training attributes. If the attributes in the training instances are not predictive of the class variable, all the models trained with these instances will be useless, because the patterns they will find will not be related to the data generating process. The number of training attributes is also important: if it is too small, the models will not be able to approximate the process; if it is too big, the noise in the training data will increase, and the models will be more prone to overfitting it; too many attributes will also increase the chances of them finding spurious correlations.

The last data mining pitfall, and perhaps the most dangerous, concerns the incorrect evaluation of a model's predictive ability. When creating a data mining model, it is extremely important not to mix the training instances with the test instances, otherwise the results will be biased. Nevertheless, even if these datasets are kept separate, we must still be cautious when analysing the model's performance with the test data. Unless the distribution of this data matches that of the general population, we cannot extrapolate from the test results how well it will predict new unseen instances. One way to mitigate this problem is to use cross-validation, rather than a single dataset, to test the model. This strategy will gives us more meaningful results, but we might still face the same issue: if the training instances do not fully represent all the new instances that the model will need to classify later on, the cross-validation results will not allow us to ascertain its true potential, because the out-of-sample data could turn out to be completely different from the training data.

In addition to these pitfalls, there is one other recurrent problem in data mining studies that needs to be emphasised. It concerns the effect of researchers' own biases in their work. Anyone that sets out to prove a point using data mining will probably "succeed" – it is basically just a question of massaging the data and configuring the models in a way that fits the intended conclusions. By doing so, we can get a model to discover the most unbelievable patterns in the training instances, and have it show excellent accuracy in backtesting. However, these patterns will be just a fluke, and the model will be useless in practical terms. Leinweber (2007) came up with a compelling example to demonstrate this point: he "proved" that from 1983 to 1993, the butter production in Bangladesh could predict the value of the S&P 500 Index very accurately. Obviously, the fact that these two variables were correlated during that period was just a coincidence; there was not cause and effect relationship, so the pattern was worthless. This goes to show that, if researchers let their prior beliefs or intentions taint the data mining process, they will get biased results that seem to support their views, but are in fact just fabrications.

If we couple all these different hazards with the randomness in financial markets, it becomes painfully clear that the creation of a model capable of making profitable price direction forecasts in any market conditions will be an extremely difficult task, if at all possible. One should expect to face issues like:

- because financial prices are very noisy, there is a big chance that the model will overfit the training data;
- the choice of training attributes is not intuitive, since no one really knows which (if any) combination of factors is behind an instrument's supply and demand at any given time; thus, we will need to select the attributes by trial and error, i.e., by training and testing the model with different combinations of attributes, and then choosing the combination that yields the best performance; however, this strategy is prone to bias, because the selected set might only work well with the test instances;
- no matter how much training data we gather, it will never be representative of all the new data that will be generated in the future; thus, we will always be dealing with insufficient information.

Besides all of these issues, we will encounter an even bigger problem: there is no way to be certain if a model is any good. Testing the model's accuracy with a validation set or cross-validation only tells us how well it performs with that specific data; since future data might be completely different, we can never know for sure if the model is a good predictor. If it is not, we will only find out after it starts making inaccurate predictions for out-of-sample instances; at that point, it will be too late, because those forecasts will already be affecting real life decisions.

After considering all the different problems we would face, we concluded it would be too time-consuming and complicated to create the agents' data mining models by hand. Thus, we devised an automatic selection mechanism for that purpose. The aim of this mechanism is to define the ensemble of models that will compose each agent's prediction module. In order to do so, it goes through all the different model types described in Section 4.1, and trains 100 models of each type using random parameters and attributes. The number and the settings of the attributes of each model are also randomized, with a few restrictions: the minimum number of attributes is 3, and the maximum is 8, so that the resulting models are neither too simple nor too complex. The most profitable model of each type is selected, based on the performance achieved with a small set of test data. Once all the different types have been processed, the selection mechanism ends up with 31 models that it believes are reasonably good. The ensemble needs to be smaller than that, so that it can output predictions in useful time. Thus, the algorithm's next step is to pick a subset of those models that is as diversified as possible. By diversified, we mean models that consistently make different predictions for the same instances; the reason for using this criterion is simple: if two models in the ensemble always output the same predictions, then one of them is redundant. Having a set of models that always make the same forecasts would defeat the purpose of the ensemble, which is to insert some redundancy in the agents' prediction mechanisms, by allowing the best models to compensate for the poor performance of the worst models in different markets conditions. In order to select the group with the most diversified set of forecasts, the algorithm compares the predictions of the 31 models for the same test data, and picks those whose predictions differ the most from all the others. This heterogeneous set becomes the ensemble in the prediction module of the trading agent.

Clearly, this selection mechanism is not immune to the pitfalls that were previously discussed. With all likelihood, many of the models that it selects will overfit the training data, and will use attributes that bear no connection with the class. Also, the algorithm picks the best model of each type by comparing the accuracy achieved with a small set of test instances; this is not a good strategy, because this dataset will not be an appropriate sample for the full population of new unseen data. Because of this, numerous worthless models will inevitably end up in the agents' ensembles, which may affect their profitability. But we do not expect this to be a big problem. The fact that some of the predictors in the ensembles will be below par simply means that the agents' adaptability skills will be put to the test. As described in Chapter 3, the agents' architecture was designed in a way that should mitigate problems caused by inaccurate models:

- if a model is not a good predictor, the weight of its contribution to the forecasts of the ensemble will be decreased;
- if all the models in the ensemble become bad predictors, the prediction module will stop making forecasts;
- even if a model performs poorly under certain market conditions, it is possible that it will perform better when the trend or the volatility change, thus proving useful in the future;
- since the models are periodically retrained with more data, bad predictors should eventually be replace with better versions of themselves.

Regardless of these mitigating factors, there is no question that the inaccurate models will have a negative effect on the trading agents. Seeing as our automatic selection mechanism is a bit naïve, it is fair to say that our agents will not be as profitable as they could have been, had we used a better selection method – like trying to come up with the best models by hand, for example. That is the biggest drawback of this mechanism. Nevertheless, it has two major advantages that justify its usage:

- it does not require manual tweaking of the data mining models, so we do not need to worry about unintentionally "over-optimizing" them and getting biased results;
- it expedites the implementation of new trading agents, because the ensembles can be selected relatively fast.

This last advantage is very important; we want to test the architecture with as many instruments as possible, and the automatic selection strategy will facilitate the process by cutting the time needed to configure each agent. In the next section, we will start presenting the trading results of numerous agents whose ensembles where selected using this strategy.

4.4 Standalone Forex Trading Agents

Following the implementation of the USD/JPY trading agent, we used the iQuant software to create another 9 intelligent agents; each was configured to trade one of the currency pairs listed in Table 12. Just like the USD/JPY agent, these new agents were trained to open a trade every 6 hours: the first at midnight, followed by trades at 6 AM, 12 PM and 18 PM GMT. They close each trade at the end of the corresponding period, before making a new prediction and opening a new trade; it is also possible that they will close a trade before the end of the period, because of the take-profit rules in their domain knowledge modules. The 6-hour investment time frame was chosen for two reasons: first, the historical price data that we were able to gather only goes back to 2003, hence a relatively short time frame was needed, so that enough instances could be extracted from that raw data to train the models; second, by using this time frame, the times of the day when the agents must send orders to the market will not usually coincide with the release of any major reports, such as interest rate decisions or the nonfarm payrolls employment change. Currency prices may become very volatile around the time these reports are first published, which implies bigger bid-ask spreads and slippage that make it more difficult to trade profitably.

We configured the prediction modules of the 10 Forex trading agents with the following settings:

- The models in each agent's ensemble, and corresponding parameters and attributes, were selected by the automatic mechanism described in the previous section. An ensemble of seven models was picked for the prediction module of each agent; this size is a reasonable compromise between speed and redundancy: the ensembles are sufficiently small that their predictions can be outputted fast enough (at the speed required for real life trading), and are also diversified enough so as to allow the agents to adapt to changes in market dynamics. The training instances used by the selection mechanism were compiled from historical price data starting in 2003, up to December of 2006. The amount of data available varied from pair to pair, therefore the number of training instances differed: it ranged from around 3,400 for the GBP/CHF pair, to around 4,000 for the EUR/USD pair. In order to select the best model of each type, the mechanism tested them with 50 instances, corresponding to the first 2.5 weeks of January of 2007. The final composition of the 10 ensembles is presented in the appendix.

- The test data sliding window, depicted in Figure 13, was also set to 50 instances. This means that, before each forecast, the prediction modules will test their models with the last 50 instances, and use the results to calculate their profit factors; as previously described, these values will in turn be utilized to define the models' vote weights, and to decide if a model should be replaced with a newer version trained with more data.

The decision to set the sliding window to just 50 instances was based on the following reasons:

- Most financial price series exhibit conditional heteroskedasticity (Franses, 1998); this signifies that the volatility is clustered, with long periods of low volatility usually being followed by short periods of high volatility. Since the weights of the models' votes are based on their profitability with the test instances, the test dataset needs to be relatively small, so that these weights can change quickly when the market enters a period of high volatility. In other words, the shorter the test set, the faster the agents will adapt to changes in market conditions.
- The sliding window method specifies that the new instance that becomes available at the end of each trading period will be used as a test instance, while the oldest instance in the test set becomes a training instance. This implies that, the shorter the test set, the faster

the newer instances will be utilized to train the models; hence, the shorter the test set, the faster the agents will be able to learn the newest patterns.

These two reasons clarify why it is generally a good idea to define a small sliding window for the test data. However, if it is set too small, the weights could become erratic – the smaller the test dataset, the more vulnerable the weights will be to the outliers in the data, which might make them less reliable. Using a sliding window with 50 instances seemed like a good compromise; it meant that, for each prediction, the weights of the models' votes would be based on their profitability in the preceding 2.5 weeks of trading.

The agents' empirical knowledge modules, responsible for suggesting the amount to invest in each trade, were configured with the following settings:

- After retrieving the cases from the database and calculating their profit factor, if that profit factor is less than or equal to 0, the suggested size for the prospective trade is 0; that is to say, if previous similar trades were mostly unprofitable, the new trade should not be opened.
- If the profit factor is between 0 and 1.5, the suggested trade size is half the user-defined standard amount.
- If the profit factor is greater than or equal to 1.5, the suggested trade size is the standard amount; thus, the agents will only invest the maximum quantity permitted when past similar trades show considerable profit.
- At least 3 cases are needed to calculate the profit factor; if the case-based reasoning system does not find enough similar trades in the database, it will retrieve the cases again, using less restrictive conditions (specifically, it will remove the last model's prediction from the information to match, and then repeat the search).

The final step in the implementation of the Forex trading agents was the configuration of their domain knowledge modules. This was accomplished with the following set of rules:

- Do not trade if it is Christmas Day, New Year's Day or Good Friday; this rule is necessary to prevent the agents from trading in low liquidity days, when there are fewer traders in the market, and thus the prices are more prone to erratic behaviour.
- Skip the first and the last trades of the week, i.e., the trades on Sunday at 18 PM and on Friday at 18 PM GMT; the reason for avoiding these trading periods is that the Forex market is less liquid during those times, which implies trading will be more costly (due to bigger bid-ask spreads and slippage) and more dangerous (because the volatility might increase more easily).
- Do not open a new trade if there is already a trade open with the exact same settings (i.e., the same direction and size); put another way, if at the end of the trading period there is already a trade open providing the exposure that the agent wants for the following period, then it should keep that trade open, instead of closing it and opening a new one; this rule is very important, because it eliminates the costs associated with the redundant trades.
- Close a trade if it reaches a profit equal to 2/3 of the average price range (in percentage) in the last 5 periods; more concretely, the take-profit target for period i is calculated with the following equation (with n set to 5):

$$tp(n)_i = 100 \times \frac{\sum_{t=1}^{n} \left(\frac{max_{i-t} - min_{i-t}}{open_{i-t}} \right)}{n} \times \frac{2}{3} \tag{9}$$

where min_{i-t}, max_{i-t} and $open_{i-t}$ are the minimum, maximum and opening prices in period $i - t$. The reason for using this equation is that it allows the agents to set profit targets according to the most recent price volatility: the target increases when the prices are more volatile, and decreases when the prices are more stable. This strategy is useful because, as we stated before, financial price series often exhibit time-clustered volatility; ergo, it makes sense to raise the target when the volatility starts increasing, because we expect it to remain high; conversely, it makes sense to decrease the target as the volatility drops. To ensure that the profit target is never too small to offset the trading costs, the actual value that the agents use in their take-profit orders is given by the maximum between 0.15% and the value calculated with Equation 9.

Once the implementation of the Forex trading agents was concluded, we had them simulate trades for the period between February of 2007 and the first half of May of 2009, which corresponds to exactly 2,510 out-of-sample instances. Their cumulative returns throughout this period are displayed in Figures 62 and 63; the full simulation results are summarized in Table 13. Considering these results, we are inclined to conclude that the Forex market is not entirely efficient. All the agents achieved a positive return at the end of the simulation period, and their average accuracy predicting the direction of the exchange rates was well over 50%; both of these accomplishments would be unlikely in a completely efficient market.

Overall, we can say that the agents' performance was impressive; their RMD ratios in particular show that they were able to trade profitably without taking too much risk. At first sight, these results seem to vindicate the usefulness of the trading agent architecture described in Chapter

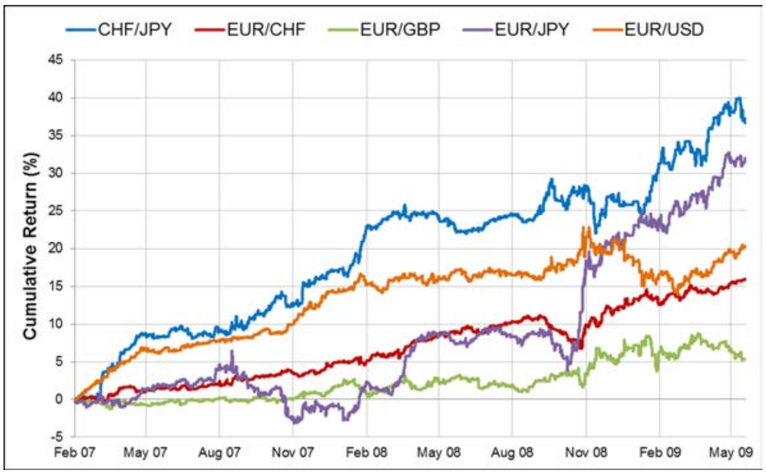


Figure 62. Gross cumulative returns of the CHF/JPY, EUR/CHF, EUR/GBP, EUR/JPY and EUR/USD trading agents.

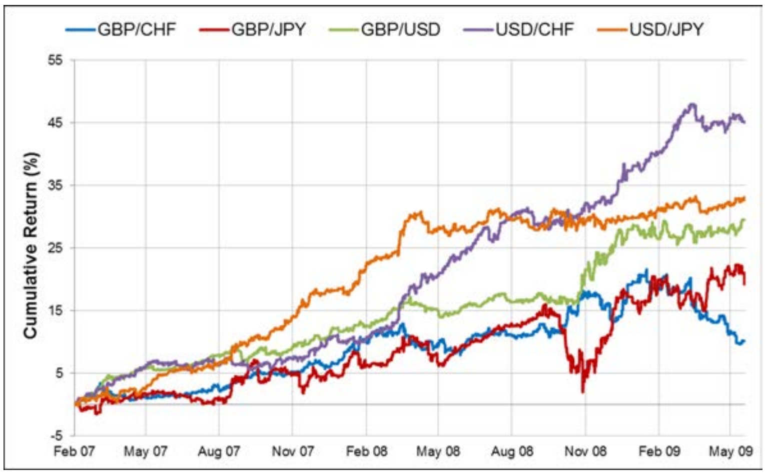


Figure 63. Gross cumulative returns of the GBP/CHF, GBP/JPY, GBP/USD, USD/CHF and USD/JPY trading agents.

3. But things are never that simple. Successful financial trading is not an easy feat, so it would be naïve to expect all the agents to trade profitably in real life. From a practical point of view, these simulation results are simply too good to be true. The pitfall here is that the statistics in Table 13 do not account for the trading costs, the "nemesis" of short term investment strategies. Hence, while we can infer from the agents' performances that they were capable of finding useful patterns in the price data, we cannot assert that they would be able to use those patterns to trade profitably in the real markets. Looking at their small returns per trade, it is clear that the trading costs would have a very negative impact on their profitability.

In order to perform a more realistic evaluation of the agents' potential, we defined a fixed cost per trade, and recalculated their cumulative returns. We tried to base this cost on actual real life trading expenses. This, however, is not straightforward, because trading commissions vary significantly from one intermediary to the next. Forex market makers, for example, do not charge an explicit commission; instead, their fees are bundled in the bid-ask spread of the currency pairs. The spread is the difference between the ask price, i.e., the price at which they are willing to sell the financial instrument, and the bid price, i.e., the price at which they are willing to buy it. Market makers usually increase the spreads when the market is less liquid, therefore the trading costs can

Table 13. Simulation results of the 10 Forex trading agents (excluding trading costs).

Agent	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Success (%)*	*Trades*
CHF/JPY	36.7	7.2	5.09	0.0322	53.6	56.0	1,139
EUR/CHF	16.0	4.4	3.64	0.0136	53.7	56.1	1,178
EUR/GBP	5.4	4.7	1.14	0.0057	49.8	52.4	937
EUR/JPY	32.0	9.6	3.32	0.0285	52.8	55.2	1,122
EUR/USD	20.4	8.9	2.30	0.0165	53.2	57.3	1,232
GBP/CHF	10.2	11.9	0.85	0.0096	51.5	55.1	1,060
GBP/JPY	19.2	14.0	1.38	0.0188	53.4	55.3	1,021
GBP/USD	29.5	3.8	7.82	0.0268	54.8	57.9	1,102
USD/CHF	45.1	4.5	9.98	0.0383	55.6	58.8	1,176
USD/JPY	32.7	3.8	8.57	0.0285	53.8	56.0	1,146

change from one trade to the next. When liquidity is high, spreads are typically very tight; for instance, one of the biggest online Forex market makers offers spreads ranging from 0.9 pips for the EUR/USD pair, to 3.9 pips for the GBP/CHF. Unlike the market makers, Forex brokers charge an explicit commission per trade. They still show a spread between the bid and the ask, but this spread depends on the supply and demand in the market, rather than being artificially imposed. For this reason, brokers' spreads in general are smaller than those offered by market makers. The fees they charge are relatively inexpensive; for example, at one of the biggest online discount brokers, the commission per trade is 0.4 pips or less (depending on the size). Since we wanted to account for other hidden costs, such as slippage and volatile spreads, we decided to test the agents with a more expensive fee of 5 pips per trade. The agents' results in the simulation period, assuming these trading expenses, are presented in Figures 64 and 65, and summarized in Table 14. While we were expecting a substantial deterioration in their performances, the end results were even worse than we imagined. Out of the 10 agents, only four were profitable at the end of the simulation period, but with negligible returns and small RMD ratios. As for the other six, they all did very poorly. The absolute worst was the EUR/GBP trading agent, which managed to lose money continuously throughout the simulation. The fact that this agent was the most affect by the trading costs should not come as a surprise, given that it had the smallest return per trade before expenses (note that this type of analysis is the reason why we chose the return per trade as a performance metric – it gives us a way to assess the agents' vulnerability to increases in trading expenses).

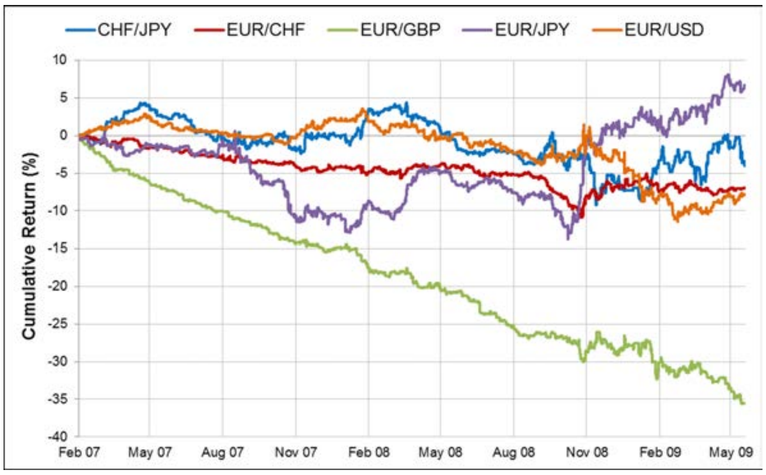


Figure 64. Net cumulative returns of the CHF/JPY, EUR/CHF, EUR/GBP, EUR/JPY and EUR/USD trading agents.

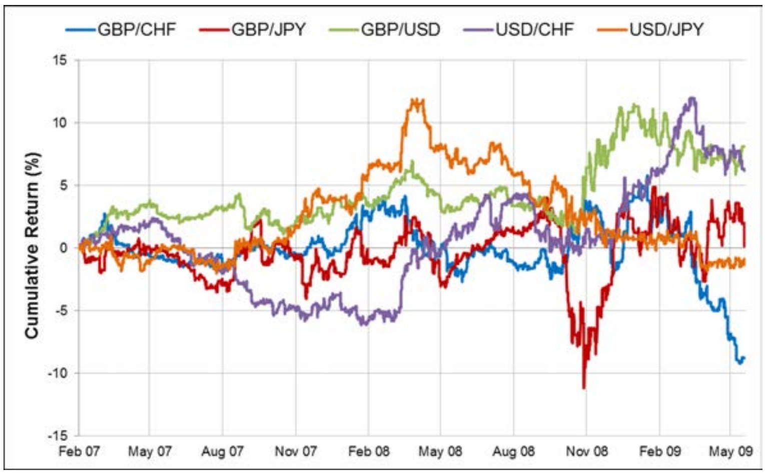


Figure 65. Net cumulative returns of the GBP/CHF, GBP/JPY, GBP/USD, USD/CHF and USD/JPY trading agents.

Considering all the metrics, we cannot find one single agent that performed acceptably. This might raise some doubts regarding the usefulness of the proposed agent architecture, at least from a practical point of view. However, if we look at the results more carefully, we can verify that the agents' dismal track records were not entirely their fault. Let us examine what happened to the EUR/GBP trading agent; the drop in its percentage of profitable trades, from 52.4% to 42.8%, hints at the real problem. Suppose the EUR/GBP exchange rate is 0.7500; the 5 pips commission we defined (i.e., 0.0005) represents a cost of around 0.07%, which seems pretty inexpensive. However, the average EUR/GBP open-to-close price variation in the 6-hour simulation periods was just 0.20%. This signifies that, as soon as the agent opened a trade, it was already spending more than one third of what it might expect to gain with that trade. It is nearly impossible to be profitable in these circumstances, no matter how skilled the trader is. The drop in the agent's success rate indicates that many of its trades that were profitable before expenses resulted in a loss once the costs were accounted for. Thus, even though the agent was "right" when it opened those

Table 14. Simulation results of the 10 Forex trading agents (including trading costs).

Agent	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Success (%)*	*Trades*
CHF/JPY	-4.0	13.6	-0.29	-0.0035	53.6	50.4	1,139
EUR/CHF	-6.9	11.0	-0.63	-0.0059	53.7	49.7	1,178
EUR/GBP	-35.5	35.5	-1.00	-0.0379	49.8	42.8	937
EUR/JPY	6.7	14.4	0.47	0.0060	52.8	52.5	1,122
EUR/USD	-7.8	15.0	-0.52	-0.0063	53.2	52.5	1,232
GBP/CHF	-8.8	15.0	-0.59	-0.0083	51.5	52.1	1,060
GBP/JPY	0.2	15.2	0.01	0.0002	53.4	52.7	1,021
GBP/USD	8.1	5.7	1.41	0.0074	54.8	53.9	1,102
USD/CHF	6.2	8.5	0.73	0.0053	55.6	52.3	1,176
USD/JPY	-1.3	13.7	-0.10	-0.0011	53.8	51.4	1,146

trades (i.e., its direction prediction was correct), it still lost money in them, because the variation of the price was too small to offset the fees. By comparing the success rates before and after subtracting the trading costs, we confirm that all the agents experienced the same problem. Clearly, configuring them to trade with a 6-hour time frame was a very bad idea, because the trading periods are too short, and hence the price changes from open to close are too small compared to the cost per trade. The implication here is that the agents' lacklustre performances were not due to their architecture, but rather to that poorly thought out decision. Therefore, we can conclude two things:

- going by stats in Table 13, the Forex agents are competent traders;
- in practice, their skills are useless, because they were configured in a way that does not addresses all the pitfalls of real life trading.

We should point out that, trading live, it is likely that the agents would end up paying considerably less than 5 pips per trade. Most discount brokers charge substantially less than that, and the tendency is for the costs to drop even further as the retail market matures. The smaller expenses would certainly improve the agents' return and overall performance. Nevertheless, before allowing the agents to trade in the real market with real funds, we believe it would be imperative to train them with a bigger time frame. Doing so should increase the range of the price in each trading period (meaning a higher potential profit per trade) hence the agents would be less affected by the trading costs. While this would definitely be the best option, we decided to attempt to improve the performance of the agents without changing their time frame. The strategies we came up with will be presented in the next two sections.

4.5 Diversified Forex Investment Strategy

As we pointed out in Chapter 3, it is possible (although not very probable) that the positive gross returns of the USD/JPY and ADBE trading agents during the simulation period were just random occurrences, or the result of specific market conditions throughout that period. We also saw that, even if their drawdowns were relatively small, both agents experienced significant losses whenever there were sudden changes in the trend or the volatility of the price. These findings indicate that we cannot blindly trust any agent, no matter how successful it was in backtesting, because there is no guarantee that it will not experience significant losses in the future. For this reason, it would be unwise to use just one agent to trade real funds. That would be the equivalent of "putting all the eggs in one basket", which is definitely not the safest or the smartest way of investing. Fortunately, there is a time-proven strategy for overcoming this problem; it is known as investment diversification, and consists in investing in various uncorrelated financial instruments simultaneously, to decrease the risk of owning each of them separately. The reasoning behind this strategy is simple: losses incurred while trading some instruments will be compensated by gains obtained while trading others, and this should yield a smoother overall return with smaller drawdowns. By making the 10 Forex agents share the monetary resources (Barbosa & Belo, 2009c), we can easily implement this type of strategy. In order to do so, we just need to evenly distribute the trading capital between them, and ensure that all their losses and gains are credited in the same brokerage account. Compared with the individual results of the 10 agents, the cumulative return of the diversified strategy should be considerably less volatile, which makes it safer in the long run.

Figure 66 displays a graphical representation of this diversified investment strategy; for clarity's sake, only 4 agents are depicted in the figure. The agents are spread across multiple hosts, so that they can make predictions and open trades faster. They use the same brokerage account to interact with the market, and share the monetary resources in that account. There is no communication between the agents, so they are not aware of each other. The simulation results obtained with this diversified strategy, using the 10 Forex agents, are presented in Figure 67 and Table 15. The chart with the cumulative return shows that, disregarding the trading costs, the performance of the diversified strategy was close to ideal: the return curve has a pronounced positive slope, which

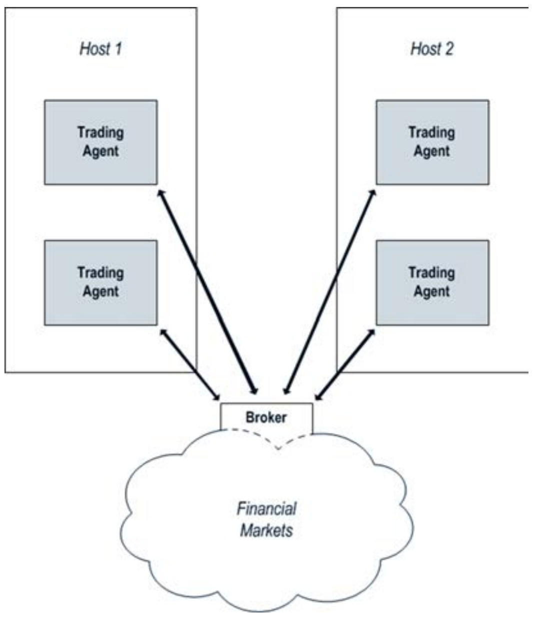


Figure 66. Graphical representation of an agent-based diversified Forex investment strategy.

indicates it was very profitable, and more importantly, this curve is very smooth, meaning the strategy was virtually risk free during the simulation period. We can confirm this analysis by looking at the stats in Table 15: the final return was 24.7%, while the maximum drawdown was an almost negligible 2.1%. The strategy's RMD ratio of 12.03 was much better than what any of the agents achieved individually (as seen in Table 13). By averaging the performances of the 10 agents, the diversified strategy was not as profitable as the best agent, and not as unprofitable as the worst; its real advantage lies in the fact that, by allowing the losses of the worst agents to be offset by the gains of the others, it experienced a much smaller maximum drawdown, without sacrificing too much profit. This improvement is exactly what we were hoping to attain. Our results confirm that investment diversification is an invaluable risk management strategy.

Much to our disappointment, when the trading costs are taken into account, the diversified investment strategy does not yield a positive return at the end of the simulation period. Overall, its performance was still better than that of the individual agents: its maximum drawdown was smaller, and it did a reasonable job covering up the losses of the worst agents. Nevertheless, its final return was negative (-4.3%), and that is all that really matters from a practical point of view. If we compare the return curves in Figure 67, it becomes clear that the agents wasted too much money on commissions; the problem lies in the high number of trades: with over 11,000 being made, the trading costs quickly add up. Something must be done about these excessive costs, if one is to develop a safe and profitable trading system that is good enough to be deployed in real life. We will look into this in the next section.

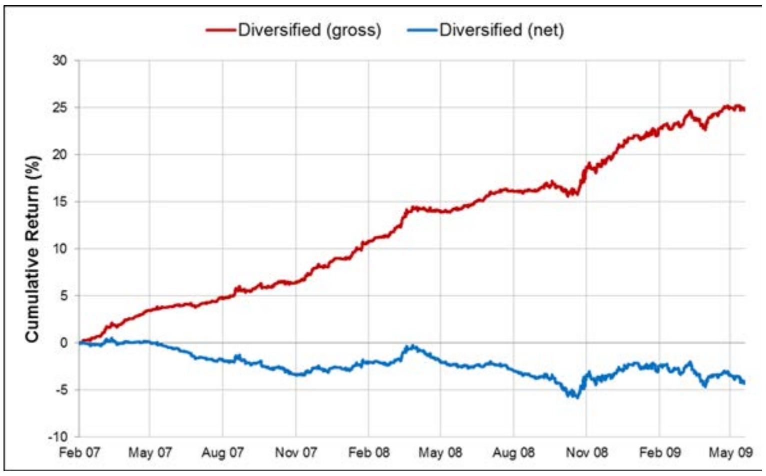


Figure 67. Gross and net cumulative returns of the diversified Forex investment strategy.

Table 15. Simulation results of the diversified Forex investment strategy (excluding and including trading costs).

Strategy	*Ret (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Acc (%)*	*Success (%)*	*Trades*
Diversified (gross)	24.7	2.1	12.03	0.0022	53.3	56.1	11,113
Diversified (net)	-4.3	6.4	-0.68	-0.0004	53.3	51.2	11,113

4.6 Multi-Agent Forex Trading Strategy

When we first started designing the trading agent architecture, we specified that the agents would be evaluated according to their RMD ratio (a pain-to-gain measure of how much profit they make per unit of risk) and their return per trade (a measure of their sensitivity to the trading costs). In the previous section, we established that investment diversification can be employed to improve the agents' overall RMD ratio. However, net results showed that we still had to increase their return per trade, in order to achieve profitability. Obviously, this can be accomplished by either increasing the total return, without increasing the number of trades, or by decreasing the number of trades required to obtain the same profit. As mentioned before, expanding the investment time frame is a possible solution, because it should allow the agents to get more profit per trade; however, this would force us to retrain all the agents with different instances. The alternative would be to come up with a way for the agents to make the same return, but with less trades. By capitalizing on the specificities of the Forex market, we were able to devise a strategy that does exactly that. In Chapter 1 we explained that whenever an agent buys a currency pair, it is in fact buying the base currency and selling the quote currency; when it shorts the pair, it is actually selling the base and buying the quote. For example, if the EUR/USD trading agent buys $100,000 of its pair, and the EUR/USD price is 1.3990, its market exposure will be long €71,500 and short $100,000. If, at the same time, the USD/JPY agent buys $100,000 of its pair, and the USD/JPY price is 89.90, its exposure will be long $100,000 and short ¥8,995,000. If we combine the market exposures of the two agents, the result is long €71,500 and short ¥8,995,000. The exact same exposure could be obtained by simply buying $100,000 of the EUR/JPY pair; therefore, in this particular situation, two trades could be replaced with just one. The unleveraged capital required for obtaining the desired exposure would also be cut in half, from $200,000 to $100,000. Now suppose the EUR/JPY agent predicts a price decrease, and short sells its currency pair with a trade size of $100,000. Considering the trades of the three agents, we are faced with the following scenario: the EUR/USD agent expects the price of the euro to increase in comparison with the U.S. dollar, the USD/JPY agent expects the price of the U.S. dollar to increase compared to the Japanese yen, and the EUR/JPY agent expects the price of the euro to decrease versus the Japanese yen. There is an obvious contradiction in these forecasts. If the prices of the three pairs actually moved in the predicted directions, this would create a glaring triangular arbitrage opportunity; the Forex market is efficient enough not to allow these short-lived profit opportunities to happen often, if ever. If we add the exposures of the three agents, we will verify that, even though there are three trades open, there is no real exposure to the market. The three trades effectively cancel each other out, so the sum of their returns will always be zero, regardless of the variations in the exchange rates. Hence, a perfect replacement for these three trades would be not to open any trades at all; this way, the agents would not pay any fees, and would not tie up any capital. This example demonstrates that, as is, the currency trading agents in our diversified investment system are making a lot of redundant trades.

As outlined in the previous section, each agent will make an investment decision concerning its currency pair at the beginning of each trading period. There are five different possibilities for this decision: buy the standard, user-defined amount; buy half the standard amount; do not trade; short half the standard amount; and finally, short the full standard amount. This means that, for every period, there are 9,765,625 ways of combining the decisions of the 10 Forex agents (that is the total number of permutations with repetition, or 5^{10}). Most of these decision combinations can be transformed into a smaller set of trades that provide the exact same market exposure. We devised an algorithm that does that conversion, and will describe it with an example. Imagine the standard trade size was set to $100,000, and the agents' decisions for a given 6-hour period were:

- Short sell $100,000 of CHF/JPY;
- buy $100,000 of EUR/CHF;
- do not trade the EUR/GBP;
- short sell $50,000 of EUR/JPY;
- short sell $50,000 of EUR/USD;
- buy $100,000 of GBP/CHF;
- short sell $100,000 of GBP/JPY;
- short sell $100,000 of GBP/USD;

- short sell $100,000 of USD/CHF;
- short sell $50,000 of USD/JPY.

If considered separately, these decisions would result in nine trades being opened, with an unleveraged capital requirement of $750,000. In order to minimize the number of trades, we start by calculating the market exposure that each of them would create:

- -$100,000 in CHF and +$100,000 in JPY;
- +$100,000 in EUR and -$100,000 in CHF;
- $0 in EUR and $0 in GBP;
- -$50,000 in EUR and +$50,000 in JPY;
- -$50,000 in EUR and +$50,000 in USD;
- +$100,000 in GBP and -$100,000 in CHF;
- -$100,000 in GBP and +$100,000 in JPY;
- -$100,000 in GBP and +$100,000 in USD;
- -$100,000 in USD and +$100,000 in CHF;
- -$50,000 in USD and +$50,000 in JPY.

Next, we add up all the exposures, to calculate the total exposure per currency:

- $0 in EUR;
- $0 in USD;
- $300,000 in JPY;
- -$200,000 in CHF;
- -$100,000 in GBP.

Now that we have determined the overall market exposure corresponding to the decisions of the 10 agents, we need to compute the smallest set of trades that will give them this exposure. In order to do so, we start by picking the currencies with the biggest positive and the biggest negative exposures. Here, they are the $300,000 in JPY and the -$200,000 in CHF. The smaller of the two, in absolute value, is the size of the first trade; in this case, it is $200,000, hence the first trade must generate an exposure of $200,000 in JPY and -$200,000 in CHF; the remaining $100,000 of JPY exposure is saved for the next iteration of the algorithm. The trade that produces the required exposure is either buying $200,000 of JPY/CHF or short selling $200,000 of CHF/JPY. Thus, the CHF/JPY agent will be responsible for short selling $200,000 of its pair. Next, we repeat the same step, using the total currency exposure remaining after the previous iteration:

- $0 in EUR;
- $0 in USD;
- $100,000 in JPY;
- $0 in CHF;
- -$100,000 in GBP.

This exposure is obtained by either buying $100,000 of JPY/GBP, or short selling $100,000 of GBP/JPY. Ergo, the GBP/JPY agent will be responsible for short selling $100,000 of its pair. All the exposure has now been accounted for, and so no other trades are needed. This means that our algorithm was able to transform the nine prospective trades into just two, which provide the exact same overall exposure. The required unleveraged capital also decreased, from $750,000 to $300,000. Notice that this is just one example, out of the 9,765,625 decision combinations that may occur. The method that was just described, listed in Algorithm 11, will cut the number of trades in 99.94% of those combinations, and the capital requirement in 99.87%. Clearly, there is much to be gained in enabling the agents to communicate their decisions to one another before opening any trades, and then having them use this algorithm to decide which trades should be opened. By eliminating the redundant trades, they will save a lot of money on fees and other trading expenses, which is precisely what they require to be more profitable in real life. All we have to do now is create an infrastructure that allows them to communicate with each other, and reach agreements before opening trades. That is to say, we must create a multi-agent system (Barbosa & Belo, 2010a). In order to do so, we defined a very simple negotiation protocol:

- whenever an agent is started, it must inform all the other agents that it will become a part of the system; likewise, it must warn the other agents before leaving the system;

```
Algorithm Agent_OptimizeForexTradeList
Inputs:
    $pair_1,pred_1,amount_1,...,pair_N,pred_N,amount_N$        // list of prospective trades to be optimized
Outputs:
    $pair_x,pred_x,amount_x,...,pair_y,pred_y,amount_y$        // optimized list of trades

BEGIN
    $curAmount_1,...,curAmount_T \leftarrow 0$                 // initialize total currency exposures
    For pair* in $pair_1 ... pair_N$                          // for each prospective trade:
        If pred* = UP Then                                    //   if it is a long trade:
            $curAmount_b \leftarrow curAmount_b$ + amount*     //     add the trade size to the total exposure of the base currency
            $curAmount_q \leftarrow curAmount_q$ - amount*     //     subtract the trade size from the total exposure of the quote currency
        ElseIf pred* = DOWN Then                              //   else if it is a short trade:
            $curAmount_b \leftarrow curAmount_b$ - amount*     //     subtract the trade size from the total exposure of the base currency
            $curAmount_q \leftarrow curAmount_q$ + amount*     //     add the trade size to the total exposure of the quote currency
        EndIf
    EndFor
    While $curAmount_1 + ... + curAmount_T > 0$                       // while there is currency exposure remaining:
        $cur_x \leftarrow$ biggest_positive($curAmount_1,...,curAmount_T$)  //  get currency with biggest positive exposure
        $cur_y \leftarrow$ biggest_negative($curAmount_1,...,curAmount_T$)  //  get currency with biggest negative exposure
        If $curAmount_x \geq | curAmount_y |$ Then                    //  if the positive is greater than the negative (in absolute value):
            $amount_P \leftarrow | curAmount_y |$                     //    size of optimized trade is the negative exposure (absolute value)
            $curAmount_x \leftarrow curAmount_x - amount_P$           //    subtract trade size from the total exposure of the first currency
            $curAmount_y \leftarrow 0$                                //    all the exposure for the second currency has been accounted for
        Else                                                          //  else if the negative is bigger:
            $amount_P \leftarrow curAmount_x$                         //    size of optimized trade is the biggest positive exposure
            $curAmount_x \leftarrow 0$                                //    all the exposure for the first currency has been accounted for
            $curAmount_y \leftarrow curAmount_y + amount_P$           //    add trade size to the total exposure of the second currency
        EndIf
        firstIsBase $\leftarrow$ is_base($cur_x$)
        If firstIsBase = TRUE Then                                    // if the currency with biggest positive exposure is the base currency:
            $pair_P \leftarrow$ create_pair($cur_x,cur_y$)            //    use it as the base currency in the pair of the optimized trade
            $class_P \leftarrow$ UP                                   //    the new optimized trade is a long trade
        Else                                                          // else if it is the quote currency:
            $pair_P \leftarrow$ create_pair($cur_y,cur_x$)            //    use it as the quote currency in the pair of the optimized trade
            $class_P \leftarrow$ DOWN                                 //     the new optimized trade is a short trade
        EndIf
        optimizedTrades $\leftarrow$ add_to_list($pair_P,class_P,amount_P$) // add the new optimized trade to the list
    EndWhile
    RETURN optimizedTrades                                            // return list of optimized trades
END
```

Algorithm 11. Pseudocode for reducing a set of Forex trades into an optimized set with equivalent currency exposure.

- after an agent makes an investment decision for a given trading period, instead of opening the corresponding trade, it communicates this decision to all the other agents;
- once an agent receives all the trading decisions, it uses Algorithm 11 to compute the smallest set of trades that can produce the desired overall exposure; if its currency pair appears in the computed set, it opens the corresponding trade, otherwise it just waits for the next period.

This protocol is depicted in the UML sequence diagram shown in Figure 68. At any given point in time, all the agents know which other agents are in the system. When the time comes to open a new trade, they make their predictions, and cooperate with each other by communicating their intentions (i.e., the trades they plan on opening). Upon receiving all these decisions, each agent will compute the smallest set of trades that can create the corresponding market exposure; since they all use the same algorithm with the same inputs, they will all compute the same set. Once this is done, each agent just needs to check if the computed set contains a trade involving its pair. If that is the case, it must open that trade; otherwise, it simply waits until the next trading period, when a new combination of decisions will be generated. While waiting for each other's decisions, the agents employ a timeout mechanism that allows them to keep operating, even if there is a problem in the communication infrastructure. Specifically, if the messages are taking longer than average, the agents will report the problem to the system administrator, and will go on trading as if they were alone in the system.

Rather than using a full-fledged agent communication language like the FIPA-ACL, which would have been overkill for such a simple negotiation protocol, we defined our own ad-hoc XML-based language for the agents' interaction. This language consists of just two types of messages:

- the *status* message, which the agents use to communicate their entry or exit from the system;
- the *decision* message, which they use to communicate their trading decisions.

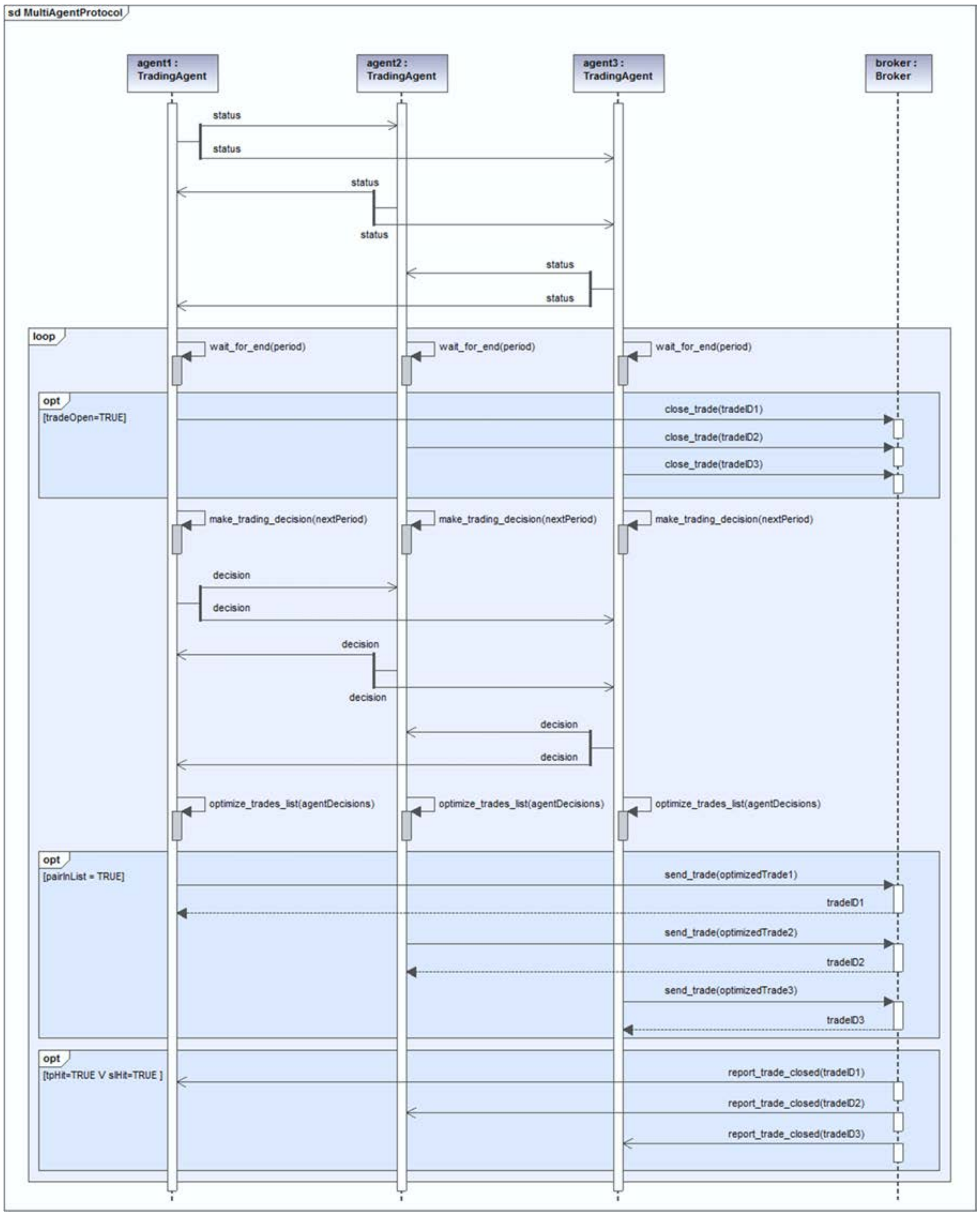

sd MultiAgentProtocol
agent1 : TradingAgent
agent2 : TradingAgent
agent3 : TradingAgent
broker : Broker
status
status
status
status
status
status
loop
wait_for_end(period)
wait_for_end(period)
wait_for_end(period)
opt
[tradeOpen=TRUE]
close_trade(tradeID1)
close_trade(tradeID2)
close_trade(tradeID3)
make_trading_decision(nextPeriod)
make_trading_decision(nextPeriod)
make_trading_decision(nextPeriod)
decision
decision
decision
decision
decision
decision
optimize_trades_list(agentDecisions)
optimize_trades_list(agentDecisions)
optimize_trades_list(agentDecisions)
opt
[pairInList = TRUE]
send_trade(optimizedTrade1)
tradeID1
send_trade(optimizedTrade2)
tradeID2
send_trade(optimizedTrade3)
tradeID3
opt
[tpHit=TRUE V slHit=TRUE]
report_trade_closed(tradeID1)
report_trade_closed(tradeID2)
report_trade_closed(tradeID3)

Figure 68. UML sequence diagram describing the negotiation protocol in the multi-agent Forex trading system.

The format of these messages is presented in Figures 69 and 70, respectively. In the *status* message, the *action* tag indicates whether the agent (stipulated in the *instrument* tag) is entering or exiting the system. In the *decision* message, the *size* tag is a code for the agent's decision: 2 for buying the standard trade size, 1 for buying half the standard size, 0 for not trading, -1 for shorting half the standard size, and -2 for shorting the full standard size.

Figure 71 shows a representation of the multi-agent Forex trading system. Compared with the simpler diversified investment strategy (depicted in Figure 66) this system adds the inter-agent communication functionality that allows the agents to minimize the number of trades made in each trading period – the lighter arrows represent the communication between them, while the black arrows represent their interactions with the market. In our implementation of the system, the communication between the agents (with the aforementioned XML messages) is handled by the ActiveMQ message broker[11]. Communication with the Forex market is carried out using the proprietary API of an online currency broker, which enables the agents to send orders to the market, receive currency price updates, and obtain information regarding the status of their trades.

```
<status>
        <instrument> EUR/USD </instrument>
        <action> IN </action>
</status>
```

Figure 69. Format of the status message.

```
<decision>
        < instrument > EUR/USD </instrument>
        <price> 1.3990 </price>
        <size> 2 </size>
</decision>
```

Figure 70. Format of the decision message.

[11] The Apache ActiveMQ message broker is available at http://activemq.apache.org/.

The system is highly scalable: the agents can be moved freely between hosts, and new hosts may be added to support a growing number of agents. Since the iQuant software was written in Java, the hosts may even be running different operating systems.

The cumulative return achieved by our multi-agent system (with the 10 Forex agents) throughout the simulation period is displayed in Figure 72; for comparison purposes, we also show the return of the simpler diversified investment strategy, in which the agents were not aware of each other. At first glance, it might seem like there is something wrong with these results. If the trading costs are not taken into account, the cumulative return of the multi-agent system should be exactly the same as the return of the simpler diversified strategy. After all, the only difference between these two strategies is that, for each trading period, the multi-agent system replaces the original set of trades with a smaller set that generates the exact same market exposure. If the exposure is the same, and there are no costs associated with the trades, then the strategies should have the same return. Yet, the chart reveals that the multi-agent strategy is much more profitable. There are two reasons for this difference. First of all, when the multi-agent system replaces a given

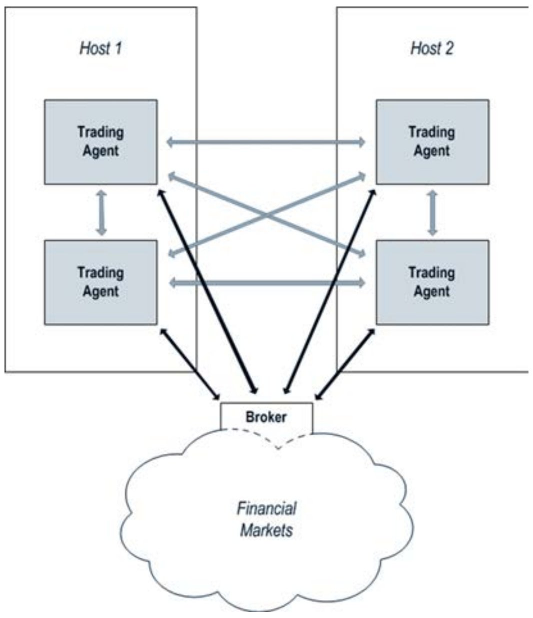


Figure 71. Graphical representation of a multi-agent Forex trading system.

set of trades with a smaller set, it just guarantees that the market exposure will be the same at the beginning of the trading period. Because we put a take-profit rule in the agents' domain knowledge modules, they can close trades at any point during that period; if this happens, the market exposure of the two strategies will no longer be the same (because the trades are different). But the hit rate of the take-profit rules is not very high, so this is not the main reason for the big divergence in the returns. What is actually making the multi-agent system perform considerably better is the fact that, in addition to lowering the number of trades, it also decreases the capital requirements. What this means, from a practical point of view, is that the agents can use a bigger trade size when they are part of the system. We will explain this feature with an example. Suppose we have \$100,000 available for investing, and do not want to employ any initial leverage. If we used a single agent to trade, we would set its standard trade size to \$100,000. On the other hand, if we used the diversified investment strategy with the 10 agents, we would need to divide that amount between them; hence, their standard trade size would be \$10,000. While we might expect to do the same when the 10 agents are part of the multi-agent system, that would in fact be a waste of resources. Going through all the 9,765,625 possible decision combinations, and the corresponding optimized sets of trades that the agents compute, we can verify that the maximum number of trades appearing

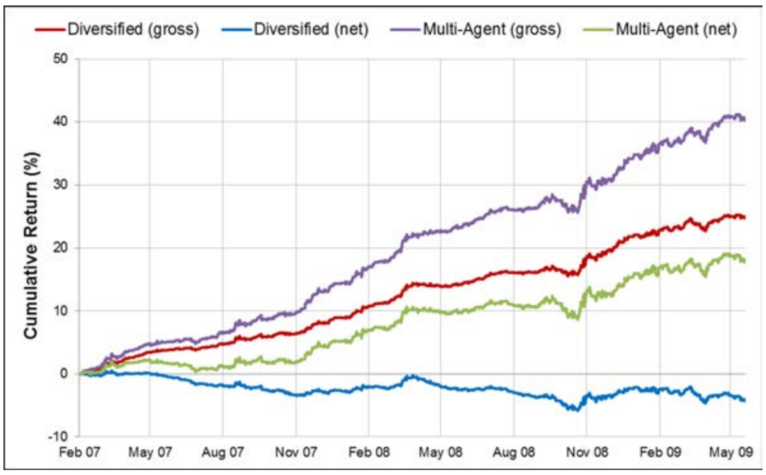


Figure 72. Gross and net cumulative returns of the multi-agent Forex investment strategy.

in any of those sets is four. Also, the maximum volume, i.e., the maximum amount of capital invested simultaneously, is only six times the standard trade size (versus 10 times for the simpler diversified strategy). Therefore, in order to maximize the utilization of the monetary resources, the standard trade size for the agents in the multi-agent system should be equal to the initial capital divided by six, or $16,667. That is almost 67% higher than the amount set for the agents in the simpler strategy. The bigger trade size signifies that, using the same money and subject to the same leverage restrictions, the agents in the multi-agent system are able to open trades with much higher market exposures. This results in bigger profits, which explains why the system achieved a better return than the simpler strategy.

The superiority of the multi-agent system is even more evident when we look at the returns after commissions. By decreasing the number of trades and the capital requirements, this system turned an otherwise unsuccessful investment strategy into a profitable strategy with low risk. The full comparison between the net results is presented in Table 16. When integrated in the multi-agent system, the agents opened less than half the trades they made when trading isolated. Consequently, they spent less money on fees, which enabled the system to reach the end of the simulation period with a positive unleveraged return of 17.8%. This is an acceptable profit after 2.3 years of trading, but far from exceptional; note that this unimpressive return is not necessarily a big issue, because in real life the system's profit may still be improved, by configuring the agents to trade with leverage – the system seems to be well suited for trading with borrowed funds, given that both its small maximum drawdown and high RMD ratio indicate that its trading strategy is not very risky. All things considered, we believe these simulation results confirm the usefulness and the potential of the agent architecture proposed in Chapter 3. In particular, they demonstrate that it may actually be possible to utilize it in the development of multi-agent trading systems safe enough to be deployed in real life.

Table 16. Comparison between the simulation results of the multi-agent system and the simpler diversified strategy (including trading costs).

Strategy	Diversified Multi-Agent	Diversified (no inter-agent communication)
Net Return (%)	17.8	-4.3
Maximum Drawdown (%)	3.8	6.4
RMD ratio	4.72	-0.68
Return/Trade (%)	0.0033	-0.0004
Trades	5,417	11,113
Volume	5,150 x standard trade size	9,176 x standard trade size
Maximum Simultaneous Trades	4	10
Maximum Simultaneous Volume	6 x standard trade size	10 x standard trade size
Standard Trade Size (unleveraged)	initial capital / 6	initial capital / 10

We should point out that, according to some stricter definitions, the described Forex trading system might be considered a simple agent-based infrastructure, rather than a true multi-agent system. Going by Panait and Luke's (2005) definition, for example, it could be argued that, since our agents are somewhat synchronized and aware of each other's states, they could be governed by a single master controller; this disqualifies our trading system from being recognized as a multi-agent system. We strongly disagree with this view. In our opinion, it should indeed be accepted as a multi-agent system, due to the following reasons:

- it is completely decentralized, and contains intelligent agents that can learn, communicate and act autonomously;
- the interaction between these self-governing entities allows them to optimize the profit of the system as a whole; nevertheless, if the communication mechanism fails, they are still able to operate on their own;

- while it is true that the agents synchronise their actions, and are aware of each other's intents, they also interact with other human and software agents of whom they have no information; this interaction occurs through the broker, whenever an agent opens a trade, because on the other side of this trade there is always a human trader or a computer acting in its own interest.

While the concept is subjective, we believe the reasons we provided should be enough to qualify the Forex trading system as a true multi-agent system.

One last thing we must emphasize is that the live trading performance of this system will greatly depend on how the agents are fine-tuned. By optimizing the way they open trades, their net return could be substantially higher. Figure 72 shows that more than half of the system's gains are still being wasted with commissions, so a decrease of just 1 pip in the average cost per trade would make a huge difference in the profit. We should be able to accomplish this decrease, by making the agents use well-timed limit orders to open the trades, instead of market orders. However, this type of optimization is beyond the scope of this thesis. Before even worrying about these details, it would be far more important to improve the level of investment diversification in the multi-agent system. As is, there are not nearly enough agents in it. Even worse, some of the Forex agents are trading instruments whose prices are often highly correlated; for instance, the EUR/JPY and the USD/JPY exchange rates frequently move in tandem. Hence, prior to letting the system trade real funds, it would be essential to insert more agents in it, that could trade other types of financial instruments, possibly with different time frames. This will be the subject of the next chapter, in which we will describe the development of stock trading agents using the iQuant software.

Chapter 5

Intelligent Agents as Autonomous Stock Traders

One of the key features of the trading agent architecture that we proposed is its versatility: it can be applied in the development of agents that trade any type of financial instrument, so long as there is historical data to train them. In particular, when the implementation is done exclusively with time and price-based attributes, all that is necessary to create new agents is some past price data, regardless of the instrument's type. This characteristic makes it extremely easy to test the architecture in different markets. In this chapter, we will study its suitability for implementing stock trading systems, by creating 25 intelligent agents with the iQuant software. Each of these agents will be responsible for trading one of the stocks listed in Table 17; the stocks in this list were picked according to three criteria:

- individually, they had to be widely traded and very liquid;
- as a whole, they needed to constitute a broad representation of the most important sectors of the economy;
- finally, we gave preference to stocks with higher beta (i.e., stocks with higher volatility compared to the rest of the market) because bigger price swings should help mitigate the negative effect of the trading costs in the agents' performances.

These 25 agents will be tested with around 3.3 years' worth of out-of-sample data. Just like in the previous chapter, we will demonstrate the importance of diversifying the investments, and the advantages of integrating the agents in a multi-agent system.

Table 17. Description of the stocks traded by the intelligent agents.

Ticker	*Company*	*Exchange*
AA	Alcoa Inc.	NYSE
AAPL	Apple Inc.	NASDAQ
ADBE	Adobe Systems Inc.	NASDAQ
BAC	Bank of America Corp.	NYSE
CAL	Continental Airlines Inc.	NYSE
CSCO	Cisco Systems Inc.	NASDAQ
DELL	Dell Inc.	NASDAQ
DIS	The Walt Disney Co.	NYSE
GE	General Electric Co.	NYSE
GOOG	Google Inc.	NASDAQ
HD	The Home Depot Inc.	NYSE
IBM	International Business Machines Corp.	NYSE
INTC	Intel Corp.	NASDAQ
JNJ	Johnson & Johnson	NYSE
KFT	Kraft Foods Inc.	NYSE
KO	The Coca-Cola Co.	NYSE
MCD	McDonald's Corp.	NYSE
MRK	Merck & Co. Inc.	NYSE
MSFT	Microsoft Corp.	NASDAQ
NVDA	NVIDIA Corp.	NASDAQ
PFE	Pfizer Inc.	NYSE
T	AT&T Inc.	NYSE
VZ	Verizon Communications Inc.	NYSE
WMT	Wal-Mart Stores Inc.	NYSE
XOM	Exxon Mobil Corp.	NYSE

5.1 Standalone Stock Trading Agents

We created 25 stock trading agents using the same method with which the Forex agents were implemented, with a few differences in the configuration. The most prominent difference is the time frame: unlike the Forex agents, which open trades every 6 hours, the stock agents were trained to trade just once a day. More specifically, every business day, each agent will open a trade at the beginning of the trading session (i.e., when the stock market opens), and close it at the end of the session (i.e., when the market closes); the trade may also be closed at any point during the day, if the take-profit rule is activated. The decision to buy or short sell the stock is made according to the agent's prediction for the direction of the price, from open to close: if it predicts the closing price will be higher than the opening price, the agent will buy the stock at the open; otherwise, if it predicts the closing price will be lower, the agent will short sell the stock at the open. The main advantage of using this time frame is that, by not keeping any trades open in-between trading days, the agents are not affected by any news published while the markets are closed (which can make stock prices gap up or down from one day to the next). On the flip side, because this is a very short time frame, it might be difficult for them to output accurate predictions, due to the noise in the price data.

The implementation of the prediction modules of the 25 agents was accomplished with the following settings:

- The automatic model selection mechanism (described in Section 4.3) picked 11 models for each agent's ensemble. Notice these ensembles are bigger than the ones of the currency trading agents; that is because those agents had to output forecasts very quickly (there is no pause between the 6-hour trading periods), so we had to keep their ensembles small, with just 7 models each. The stock agents do not suffer from this limitation, because there is a long break between the close of the market on one day, and its opening the next, which means they have several hours to retrain the models and make the forecasts. Hence, we increased the size of the ensembles to 11 models, so that their prediction mechanisms would be more diversified, and hopefully more reliable. To train these models, the selection mechanism used all the historical price data that we could gather, up to October of 2005. The actual number of training instances varied, as some stocks have been around far longer than others; it ranged from about 300 instances for the GOOG agent, to over 11,000 for the IBM agent. To test and select the models, the mechanism used the 50 instances corresponding to the 2.5 months period between November of 2005 and the second week of January of 2006. The final composition of the 25 ensembles is shown in the appendix.

- The sliding window for the test data, depicted in Figure 13, was also set to 50 instances. Consequently, the models' replacement and vote weights for each prediction are based on their profitability in the previous 2.5 months.

The empirical knowledge modules of the 25 agents were configured with the same settings as the Forex agents, only the threshold for suggesting the standard trade size was set to 1, instead of 1.5. This change was meant to increase the return of the stock trading agents, by forcing them to take on more risk – the lower threshold implies that they will invest the maximum amount more often.

Finally, the agents' domain knowledge modules were configured with the following set of rules:

- Close a trade if it reaches a profit equal to 2/3 of the average price range in the previous 5 periods (Equation 9); this rule is similar to the one utilized by the Forex agents, only the minimum take-profit target was set to 0.5%, instead of 0.15%, to accommodate for the greater costs associated with stock trading, as well as the higher volatility of stock prices.

- Do not trade if the stock's opening price is below \$10; this rule is important because cheap stocks are more expensive to trade – if commissions are too costly, the agents are better off not trading at all.

As already discussed in Chapter 4, in order to obtain realistic results, it is essential to account for the trading costs. Given that stock trading fees vary significantly, depending on the broker, there is no clear choice for the cost to emulate in the trading simulations. To test our agents, we decided to base this cost on the commissions charged by one of the biggest online discount brokers: \$0.01 per share, with a minimum of \$2 and a maximum of 1% per trade. To calculate the commissions, we assumed a standard trade size of \$20,000. Figure 73 shows the cost of a trade (as a percentage of the amount invested) for different stock prices, considering this standard trade size and the aforementioned trading fees. This chart clarifies why we had to define a rule to prevent the agents from trading when the stock prices are below \$10. Without this rule, the cost of a trade could be as high as 1%, which would make it almost impossible to trade profitably. By inserting the \$10 cut-off rule in their domain knowledge modules, we ensured that none of the agents would ever pay more than 0.1% for a trade.

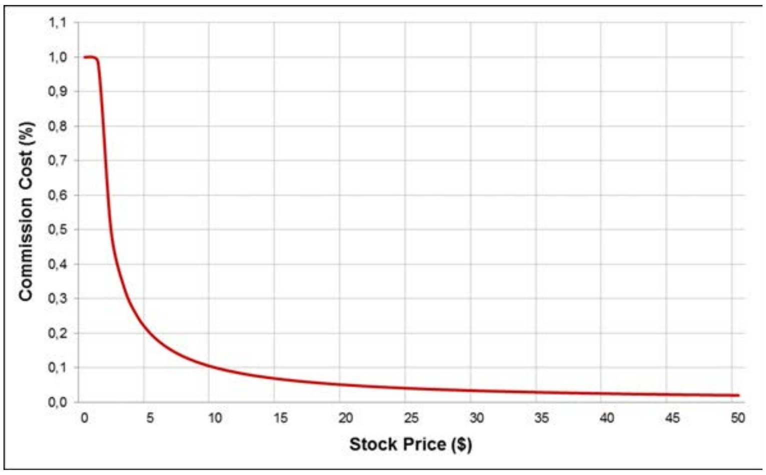


Figure 73. Trade cost for different stock prices.

Once trained and configured, the 25 agents simulated trades in the period between February of 2006 and May of 2009. This corresponds to a total of 854 test instances. The simulation results are presented in Figures 74 to 78, and summarized in Table 18.

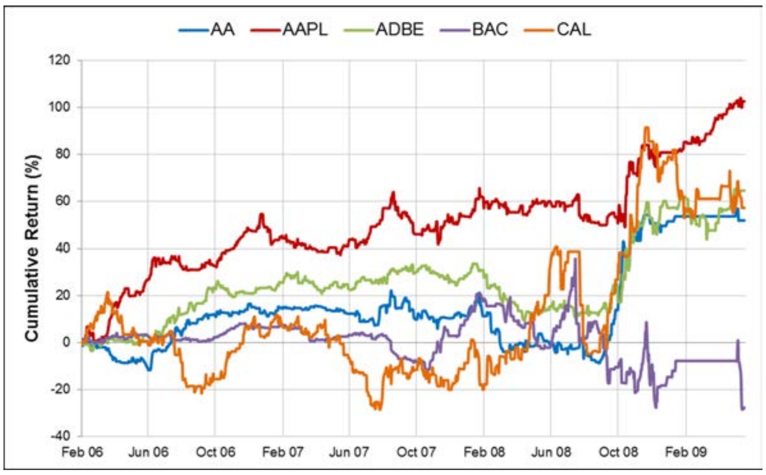


Figure 74. Net cumulative returns of the AA, AAPL, ADBE, BAC and CAL trading agents.

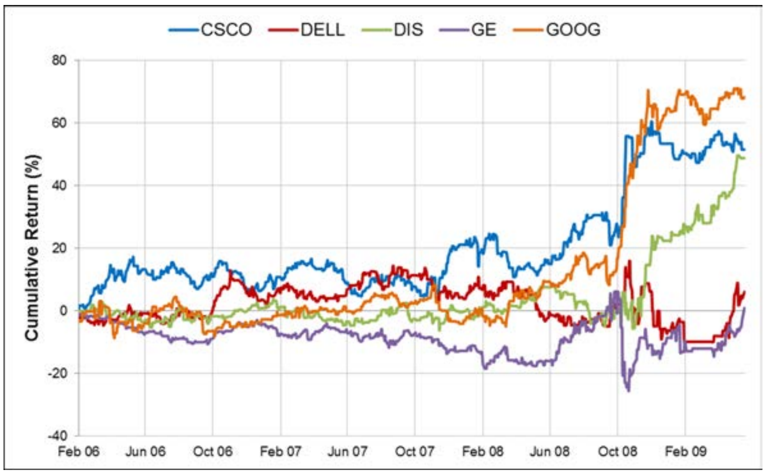


Figure 75. Net cumulative returns of the CSCO, DELL, DIS, GE and GOOG trading agents.

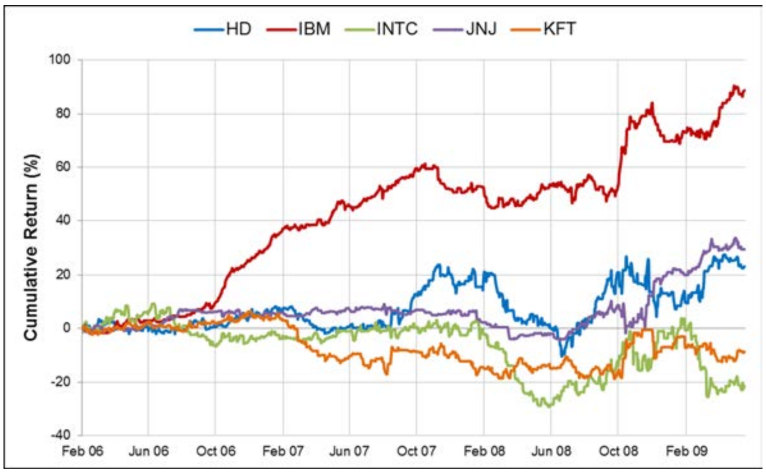


Figure 76. Net cumulative returns of the HD, IBM, INTC, JNJ and KFT trading agents.

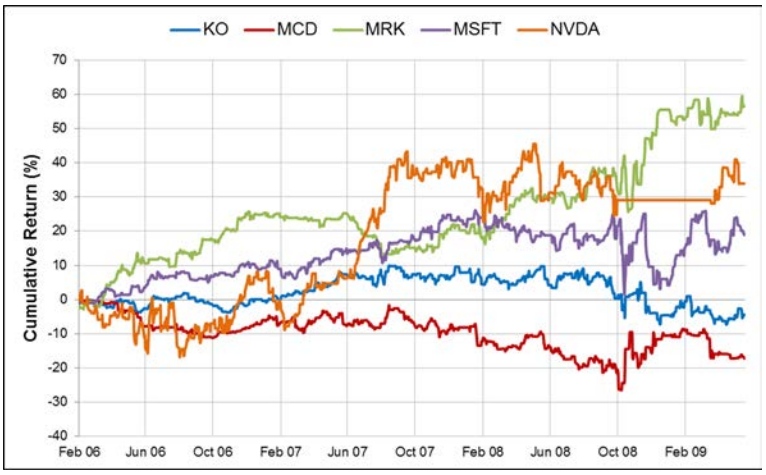


Figure 77. Net cumulative returns of the KO, MCD, MRK, MSFT and NVDA trading agents.

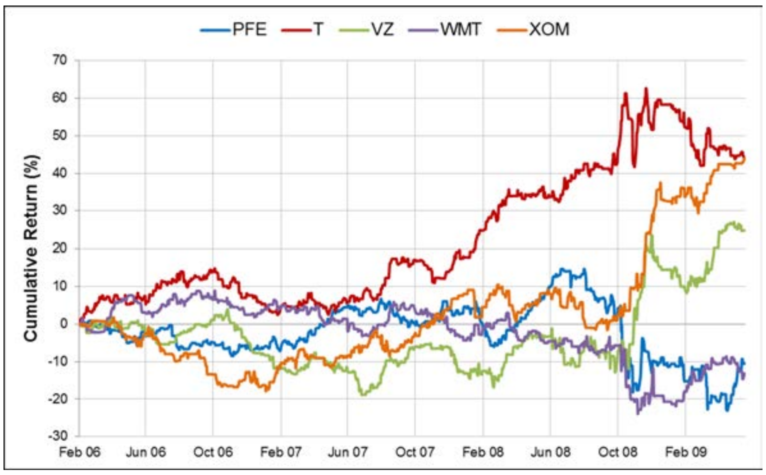


Figure 78. Net cumulative returns of the PFE, T, VZ, WMT and XOM trading agents.

Table 18. Simulation results of the stock trading agents (including trading costs).

Agent	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Success (%)*	*Trades*
AA	51.9	30.9	1.68	0.1457	49.2	51.4	356
AAPL	102.6	22.0	4.67	0.2449	57.8	60.4	419
ADBE	64.6	24.1	2.68	0.1513	51.5	54.8	427
BAC	-27.5	63.7	-0.43	-0.0680	49.6	52.8	405
CAL	57.3	50.0	1.15	0.1354	52.0	54.1	423
CSCO	51.5	13.9	3.71	0.1026	52.6	55.4	502
DELL	6.1	25.8	0.24	0.0144	49.6	52.0	423
DIS	48.8	13.6	3.58	0.1173	53.6	54.6	416
GE	1.0	31.6	0.03	0.0023	53.0	54.4	423
GOOG	68.3	13.3	5.15	0.1728	56.7	58.2	395
HD	23.2	34.1	0.68	0.0472	49.4	51.8	492
IBM	88.7	16.5	5.37	0.1677	55.2	58.6	529
INTC	-21.3	38.4	-0.55	-0.0440	50.2	51.4	484
JNJ	29.6	13.7	2.17	0.0626	51.8	55.6	473
KFT	-8.7	25.2	-0.34	-0.0189	49.0	52.5	459
KO	-4.2	17.2	-0.25	-0.0097	50.2	52.5	436
MCD	-17.1	26.4	-0.65	-0.0418	48.4	50.1	409
MRK	56.7	16.5	3.44	0.1297	49.2	53.3	437
MSFT	18.9	24.1	0.78	0.0424	53.7	56.2	445
NVDA	33.9	20.8	1.63	0.0854	55.7	57.4	397
PFE	-10.5	37.7	-0.28	-0.0219	49.0	52.5	478
T	43.9	20.6	2.13	0.0986	50.1	52.4	445
VZ	24.8	22.8	1.09	0.0561	51.4	53.2	442
WMT	-13.1	32.7	-0.40	-0.0313	47.7	50.8	419
XOM	44.0	19.7	2.24	0.1082	50.1	53.3	407

Our primary goal since the start was always to create agents that could mimic human traders. Judging by the results of the 25 stock trading agents, it seems we accomplished this objective all too well: just like in real life, some agents had an excellent performance, others did just ok, and some were just plain incompetent. The great majority had disappointing accuracy, which means that, either daily stock prices are simply too random to be predicted (and there is nothing we can do about that) or the attributes that we used to train their data mining models were not good enough. While it is undeniable that stock prices are extremely hard to predict, we do believe that researching better attributes – based on fundamentals, for example – would help us improve the accuracy of the stock trading agents, which in turn should increase their returns. That would be one way to better these agents, although there is always the possibility that the higher accuracy would not result in bigger profits.

Even if their overall accuracy was below par, that did not prevent 18 of the 25 agents from being profitable at the end of the simulation period. Looking at their RMD ratios, some of them actually showed a lot of promise. These include the AAPL, the IBM and the GOOG trading agents, among others. On the flip side, some agents were just begging to be "fired". One of the worst was the BAC agent, but its losses are somewhat understandable. Figure 79 displays the historical prices of the BAC stock, with a separation between the data that was initially used to train the agent, and the data that was utilized to simulate the trades. We can verify that the behaviour of the price during the simulation period is far from "normal", with a 93% free-fall drop (caused by the subprime mortgage crisis). While we expect the agents to be able to adapt to changes in the trend and in the volatility of the price, it would be unrealistic to expect them to make accurate predictions in such a chaotic environment. In a sense, it may even be argued that the BAC agent performed acceptably, as it avoided the enormous losses that one would incur with the buy-and-hold investment strategy; this comparison is presented in Figure 80. Unfortunately, the same reasoning does not explain the bad performances of other incompetent agents. For example,

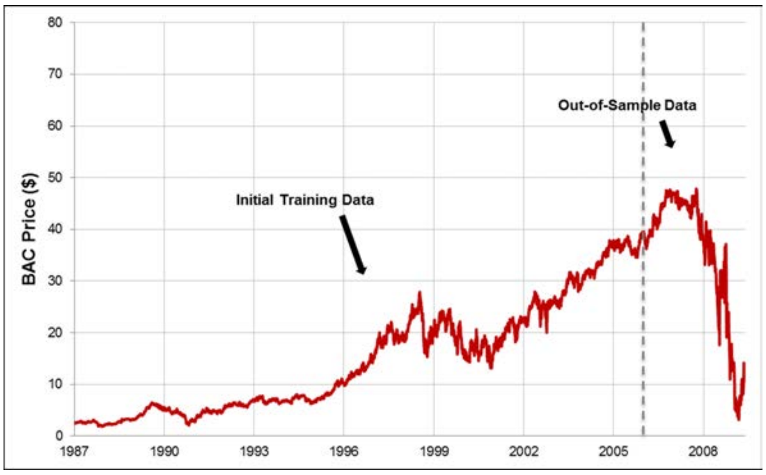


Figure 79. Historical BAC stock prices.

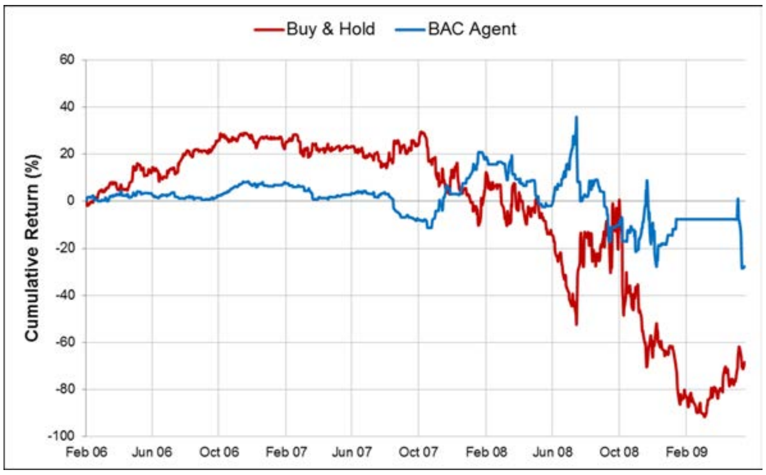


Figure 80. Net cumulative return of the BAC trading agent, compared with the buy-and-hold strategy.

the return of the MCD trading agent was much worse than what we could get, by simply buying the stock at the beginning of the simulation period, and holding it until the end, as shown in Figure 81.

One obvious conclusion that one may draw from this experiment is that the agent architecture we proposed is not infallible, i.e., it does not guarantee that every single agent will trade profitably and safely, especially if the attributes used to train its data mining models are not good predictors of the instrument's price direction. In spite of that, we believe the simulation results were, as a whole, quite positive, and supportive of the usefulness of the architecture. Most of the agents traded profitably during a particularly eventful period in the stock markets, and that is no small feat. Still, we cannot ignore the big maximum drawdowns, a red flag indicating that there is substantial risk associated with their individual strategies. This is yet another sign that it would be far too dangerous to trust a single agent to trade real funds. Luckily, our experiments with the Forex agents have already confirmed that we can eliminate much of the trading risk by diversifying the investments. We will test a new diversified strategy in the next section, based on the 25 stock trading agents.

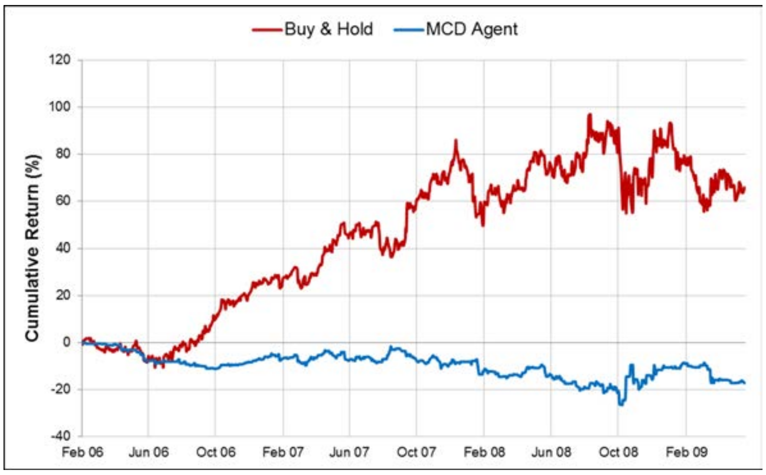


Figure 81. Net cumulative return of the MCD trading agent, compared with the buy-and-hold strategy.

5.2 Diversified Stock Investment Strategy

Black swan events, such as the subprime mortgage crisis of 2008, are relatively rare. Even so, an investment strategy needs to be resilient to these occurrences, in order to withstand the test of time. There are numerous examples of investment companies that were able to achieve high returns for several years in a row, only to go bust due to one of these events. According to the Hedge Fund Research firm (2009), 1,471 hedge funds were liquidated in 2008 alone. As we demonstrated in the previous chapter, one way to minimize the risk and mitigate the impact of these infrequent occurrences is to diversify the investments. But investment diversification is not, by itself, a guarantee of safety. The S&P 500 Index, which tracks the stock prices of 500 U.S. companies, is by definition a very diversified benchmark; yet, it still experienced a loss of over 55% between 2007 and 2009. This form of diversification is not the best, because the index only has exposure to one asset class, and all the exposure is on the long side. We implemented a diversified investment strategy with our 25 stock trading agents (Barbosa & Belo, 2010b) that should prove more reliable – even if these agents are all trading instruments of the same class, at least they can go long or short whenever they want. This system was created following the same method that was used with the Forex agents (which we described in Section 4.5). Its trading results in the simulation period are presented in Figure 82 and Table 19, in comparison with the performance of the simpler buy-and-hold strategy. These results show that, as expected, grouping the agents and dividing the monetary resources between them gave way to a relatively safe investment strategy. Its net return of 28.5% after 3.3 years of trading is acceptable, but not very impressive; still, this is not a big problem, because the strategy's low maximum drawdown indicates we could improve its return using leverage, without incurring too much risk. Overall, we can conclude that the strategy has potential.

The chart in Figure 82 reveals that almost one third of the agents' profit was spent on trading fees. While this is significant, it is not nearly as bad as what we saw with the Forex agents in Section 4.5 – there, the costs actually made the diversified strategy unprofitable. We believe this difference is partly due to the stock agents having a bigger investment time frame, which implies bigger price variations, and more potential profit per trade; the fact that stock prices are more volatile might also have helped. Despite this difference, the trading expenses are still too high, so it would be worth considering a further increase in the time frame.

Compared with the buy-and-hold strategy, the agent-based stock trading strategy did really well: the smoothness in its cumulative return curve confirms it was much safer in the simulation period, and its final return was also much better. Nevertheless, during the bull market of 2006 and

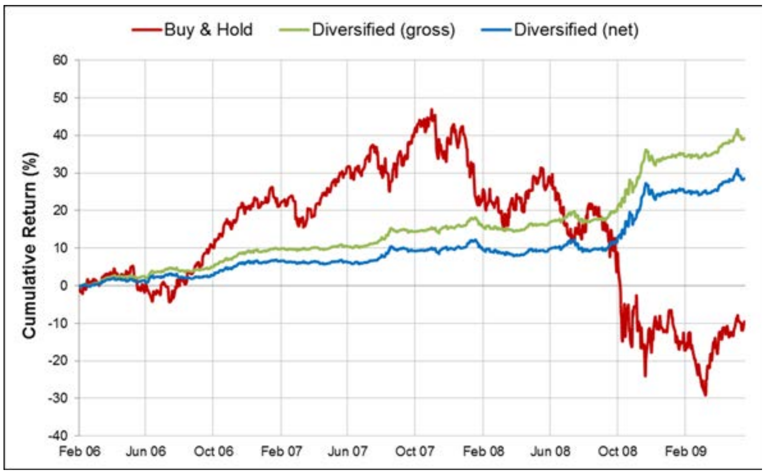


Figure 82. Gross and net cumulative returns of the diversified stock trading system, compared with the buy-and-hold strategy.

Table 19. Simulation results of the diversified stock trading system, compared with the buy-and-hold strategy.

Strategy	*Ret (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Acc (%)*	*Success (%)*	*Trades*
Buy & Hold	-9.5	51.7	-0.18	-0.3819	40.0	40.0	25
Diversified (gross)	39.2	4.2	9.35	0.0036	51.5	54.5	10,941
Diversified (net)	28.5	4.4	6.53	0.0026	51.5	54.0	10,941

2007, the buy-and-hold strategy yielded a much bigger return. This is not surprising, because stock prices usually increase very rapidly during bull markets, with few breaks; in these conditions, buying the financial assets and never selling them is the perfect way to trade. However, when things go awry, this strategy will fail miserably, and suffer enormous drawdowns. For example, buying the 25 stocks at the end of 2007 and holding for a year would have resulted in a loss of over 50% of the investment. As we see it, resilience in the toughest of times is the single most important characteristic that an investment strategy must possess. In that respect, our agent-based solution was vastly superior to the buy-and-hold strategy. But the fact that we prefer to put the emphasis on safety, rather than profit, does not preclude us from pursuing better returns. In the next chapter, we will demonstrate how trading with leverage can attenuate the difference in profit between our solution and the buy-and-hold strategy during bull markets.

One thing that stands out in the chart in Figure 82 is the fact that the cumulative return of our diversified strategy increased slowly, but steadily, regardless of whether the market was going up or down. This is a distinguishing characteristic of good investment strategies. Replicating this type of performance, where the return grows continuously without much volatility, irrespective of the direction of the market, is an extremely difficult task. We can easily create a strategy that works well in specific conditions, the difficulty is in making it perform acceptably all the time. To demonstrate this point, we replaced the agents in our solution with 25 trading bots; these bots were configured to trade using two naïve strategies: first, each bot was instructed to buy the stock if its price went up in the previous day, or short sell it if it went down; next, they were instructed to do the opposite, i.e., buy if the price went down, and short sell if it went up. During the tests, the bots were subjected to the same trading fees as our agents. The cumulative returns obtained with the two strategies are shown in Figure 83, compared with the return of the agent-based solution. Not surprisingly, the performance of the first naïve strategy was a complete disaster; the contrarian

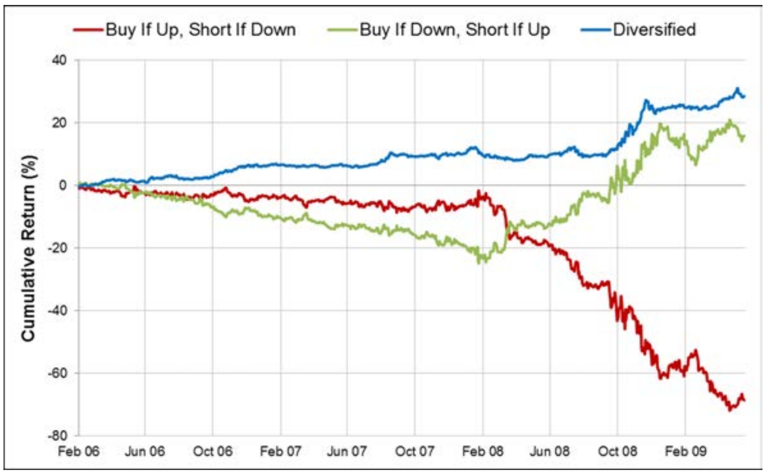


Figure 83. Net cumulative returns of two systems with naïve trading bots, compared with the agent-based diversified system.

strategy, on the other hand, did poorly while the market was trending upward, but performed quite well when it started trending downward. If we restrict our analysis to the period between February and December of 2008, this naïve strategy will seem better than our agent-based solution. Obviously, this comparison is deceiving, because during that period the market moved almost exclusively in one direction. When evaluating an investment strategy, it is vital to look at its performance in different market conditions, not just when the environment is favourable – that is the only effective way to assess its true potential to be successful in the long run. This is the reason why it is so important that the test data thoroughly represents all different scenarios. Luckily, the 3.3 years' worth of data that was used to test our agents contains both market extremes: a powerful bull market, followed by a crash of historic proportions. The fact that our diversified stock trading solution was able to perform well in both situations is undoubtedly a great achievement. It is this ability that separates it from simpler strategies, that only work in specific market conditions.

According to Table 19, the combined accuracy of the 25 agents after 10,941 trades was just 51.5%. As previously mentioned, using better attributes to train the agents should improve this figure, although we would not expect a very dramatic improvement, as there is simply too much noise in daily stock prices. Fortunately, high accuracy is not a requirement for profitable trading. Despite the low precision, the agents still obtained a combined gross return of 39.2%, or 28.5% after fees. This implies that they were capable of predicting the most important trades, i.e., the ones with the biggest price variations. Since the success rate is greater than the accuracy, we can also conclude that the agents' empirical and domain knowledge modules helped compensate for the lack of precision of the prediction modules. To verify this claim, we repeated the trading simulation using agents based on simpler architectures. First, we made them use only the prediction modules (in accordance with the architecture presented in Figure 9); next, they utilized a combination between the prediction and the empirical knowledge modules (as seen in Figure 30); lastly, they used a combination between the prediction and the domain knowledge modules (Figure 35).

Figure 84 and Table 20 present the simulation results obtained with the corresponding diversified investment strategies. These results are similar to the ones we got in Chapter 3, when we tested the contribution of each module to the performances of the USD/JPY and the ADBE trading agents. With the simpler version of the agents, which consisted of just the prediction modules, the overall accuracy of the diversified strategy was a meagre 51.0%; the return was 9.9%, with a comparatively high maximum drawdown of 11.8%. Adding the empirical knowledge module to the agents' implementation cut the number of trades in half, and led to a significant increase in both the overall RMD ratio and the return per trade. Combining the prediction modules with the domain knowledge modules resulted in an even bigger improvement in both metrics, although the number of trades did not change much. Finally, the actual stock trading agents (which employ the three modules simultaneously) achieved the best performance of all, be it in terms of RMD ratio, return per trade, accuracy, or even percentage of profitable trades. This proves that each of the three modules made an important contribution to the performance of the agents. Ergo, this empirical evidence suggests that the internal structure that we chose for the trading agent architecture makes sense not only in theory, but also in practice.

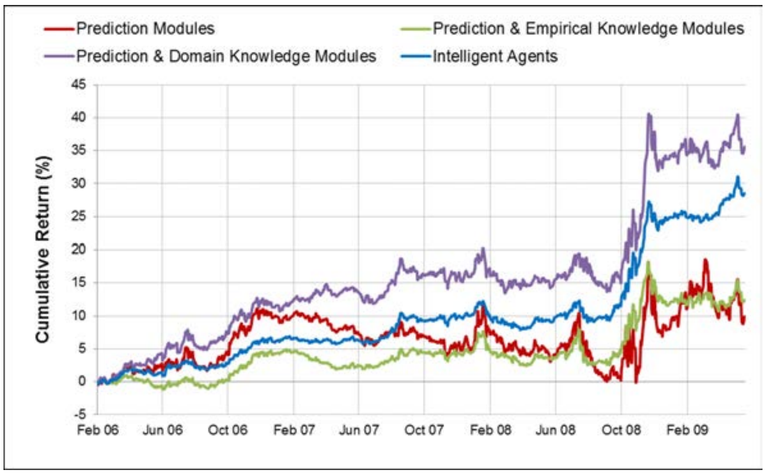


Figure 84. Net cumulative returns of the diversified stock investment system using agents based on different architectures.

Table 20. Simulation results of the diversified stock investment system using agents based on different architectures (including trading costs).

Architecture	*Ret (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Acc (%)*	*Succ (%)*	*Trades*
Prediction (Figure 9)	9.9	11.8	0.84	0.0005	51.0	50.6	20,714
Prediction + Empirical (Figure 30)	12.4	7.5	1.66	0.0011	51.4	50.8	11,307
Prediction + Domain (Figure 35)	35.6	8.7	4.11	0.0018	51.1	53.6	20,044
Intelligent Agents (Figure 40)	28.5	4.4	6.53	0.0026	51.5	54.0	10,941

On balance, after analysing the results of the diversified stock trading strategy, we arrive at the same conclusion that we did in the previous chapter: not all the agents based on the proposed architecture will be talented, but we can nonetheless use it to build promising trading systems, by combining enough agents (which should be as heterogeneous as possible).

5.3 Multi-Agent Stock Trading Strategy

In Chapter 4, we saw that there was a very good reason for integrating the Forex agents in a multi-agent system: by communicating their decisions to one another, these agents were able to eliminate numerous redundant trades (and the related costs) which significantly improved their overall return. The need for implementing this type of system with the stock trading agents is not as obvious. When an agent decides to buy or short sell a stock, there is not much that the other agents can do with that information; hence, inter-agent communication does not seem necessary. However, there is a specific scenario in which it would be important for them to report their decisions. Suppose the 25 agents were given a certain amount of euros to trade in the stock market.

Since they were trained to negotiate stocks denominated in U.S. dollars, there will be currency risk associated with their trades. If they want to buy stocks, they first need to convert the euros to dollars, and only then use the dollars to buy the securities; when they sell stocks, they need to do the opposite, i.e., convert the dollars they receive back to euros. This implies that the profitability of their trades will depend not only on the changes in the stock prices, but also on the EUR/USD exchange rate. This currency risk is not negligible; on the contrary, it could have a very big impact on the return of each trade (negative or positive). For example, imagine the ADBE agent decided to buy €10,000 of the stock, with the EUR/USD exchange rate at 1.2000. In order to open the trade, it must first buy $12,000 using the €10,000, after which it can purchase the shares with the dollars. Now imagine the price of the ADBE stock went up 1% during the trading day, while the EUR/USD price increased 1.5%. In these circumstances, the value of the shares that the agent bought will have increased to $12,120, giving it a profit of $120. However, after selling the shares, the agent needs to convert those dollars back to euros. Since the EUR/USD exchange rate at the time of the sale increased to 1.2180, the agent will only get back around 9,950 euros, which means the final return for the trade is actually -0.5%. In practice, we can say the agent made two different trades, gaining 1% in one of them (buying the stock), but losing 1.5% in the other one (buying the dollar). So, even though the agent was correct when it predicted an increase in the price of the stock, it still lost money in the trade, due to its long U.S. dollar exposure. The implication here is that the agents' success does not depend solely on their trading skills, which could be problematic over the longer term. To solve this issue, we devised a multi-agent system that eliminates the currency risk associated with stock trades. This system is composed of the 25 stock trading agents, and a special hedging agent, whose only objective is to guarantee that the system's overall currency exposure is always zero. We defined the following negotiation protocol for the system:

- whenever a stock agent is started, it must inform the hedging agent that it is entering the system; likewise, it must warn the hedging agent before exiting the system;

- after a stock agent makes a trading decision, it must report this decision to the hedging agent, and only then open the corresponding trade;
- once the hedging agent receives the trading decisions of all the stock agents, it calculates the overall currency exposure that the corresponding trades will create; then, it opens its own trades in the Forex market, to hedge that exposure;
- whenever a stock trade is closed, the hedging agent must be notified by the broker, so that it can open or close trades in the Forex market, to ensure that the system's remaining currency exposure is properly hedged.

This protocol is depicted in the UML sequence diagram in Figure 85. The process it describes is very simple: the stock agents communicate their decisions to the hedging agent, and then open the trades; the hedging agent will in turn calculate the total currency exposure generated by those decisions, and will hedge that exposure in the Forex market. For instance, if the hedging agent determines that the trades of the 25 stock agents will create a long U.S. dollar exposure, it will buy an equivalent amount of the EUR/USD currency pair (i.e., buy euros, sell dollars). In this situation, if the price of the EUR/USD increases, the stock agents will lose money due to being (unwittingly) long the dollar, but the profit of the Forex trades of the hedging agent will offset that loss. On the other hand, if the EUR/USD price drops, the stock agents will get some extra profit when they exchange their dollars for euros, but the hedging agent will lose an equivalent amount in its trades. Hence, this protocol guarantees that the return of the multi-agent system will never be affected, neither positively nor negatively, by variations in the exchange rates. Obviously, the solution we described will only be useful if some of the stocks being traded are priced in currencies other than the system's base currency.

Figure 86 displays a graphical representation of our multi-agent stock trading system; for clarity's sake, only three stock agents are represented. The lighter arrows in the graphic symbolize the communication between these agents and the hedging agent, while the black arrows symbolize

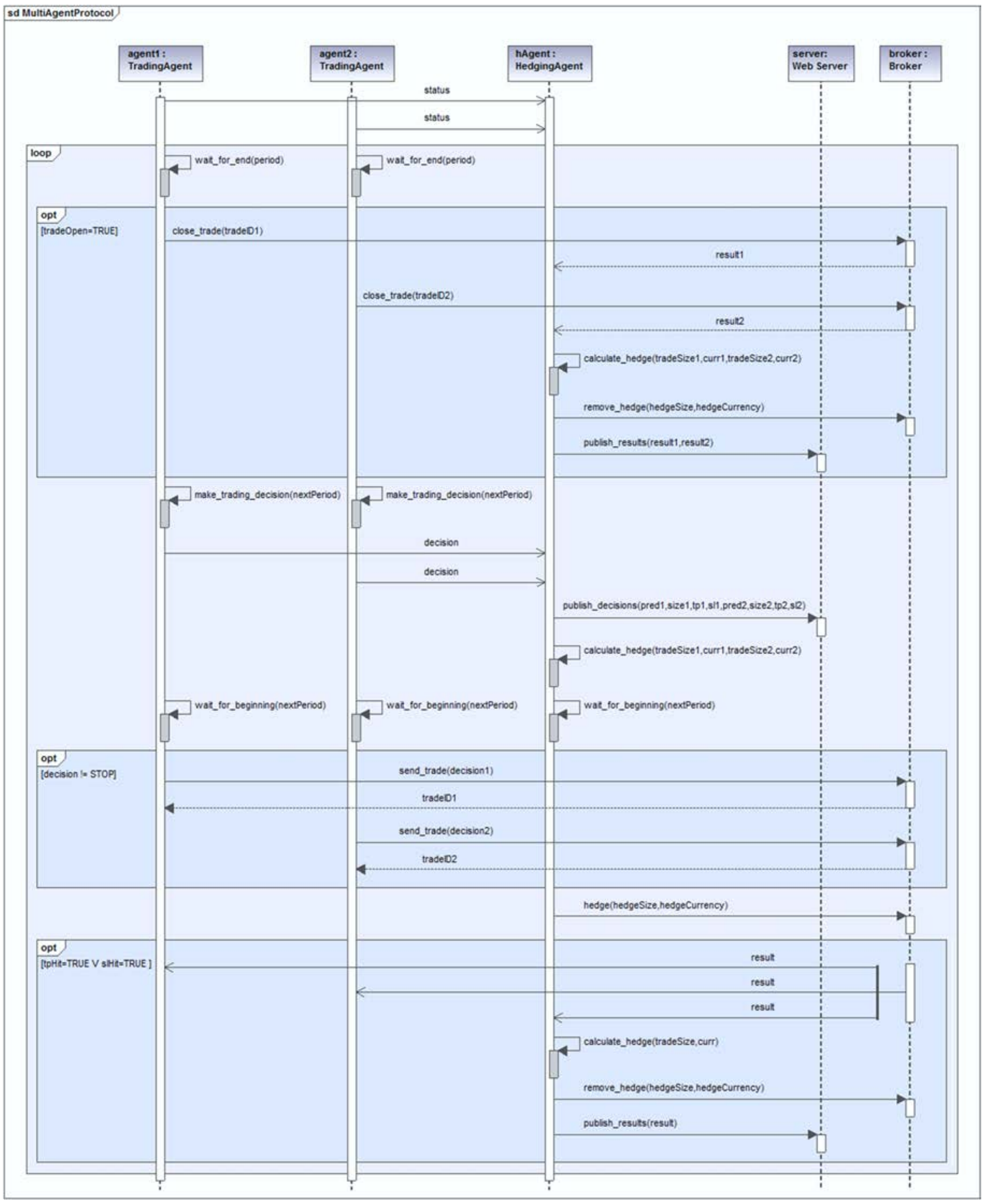

sd MultiAgentProtocol
agent1 : TradingAgent
agent2 : TradingAgent
hAgent : HedgingAgent
server: Web Server
broker : Broker
status
status
loop
wait_for_end(period)
wait_for_end(period)
opt
[tradeOpen=TRUE]
close_trade(tradeID1)
result1
close_trade(tradeID2)
result2
calculate_hedge(tradeSize1,curr1,tradeSize2,curr2)
remove_hedge(hedgeSize,hedgeCurrency)
publish_results(result1,result2)
make_trading_decision(nextPeriod)
make_trading_decision(nextPeriod)
decision
decision
publish_decisions(pred1,size1,tp1,sl1,pred2,size2,tp2,sl2)
calculate_hedge(tradeSize1,curr1,tradeSize2,curr2)
wait_for_beginning(nextPeriod)
wait_for_beginning(nextPeriod)
wait_for_beginning(nextPeriod)
opt
[decision != STOP]
send_trade(decision1)
tradeID1
send_trade(decision2)
tradeID2
hedge(hedgeSize,hedgeCurrency)
opt
[tpHit=TRUE V slHit=TRUE]
result
result
result
calculate_hedge(tradeSize,curr)
remove_hedge(hedgeSize,hedgeCurrency)
publish_results(result)

Figure 85. UML sequence diagram describing the negotiation protocol in the multi-agent stock trading system.

the interactions with the market – the stock market in the case of the stock agents, and the Forex market in the case of the hedging agent. Besides opening trades to offset currency exposures, the hedging agent is also responsible for publishing the trading decisions of all the stock agents in a public website – the iQuant website[12] – before the start of each trading session. This functionality makes it possible to follow the agents' activity in real-time. The website shows the most up-to-date performance information for the multi-agent stock trading system, in comparison with the performance of the buy-and-hold investment strategy. It also shows the individual performance of each stock agent, and the historical results that would be obtained with simpler agent architectures. The 25 trading decisions are updated daily at around 8 AM GMT, about 6 hours prior to market open, when the trades should occur. Once the trading session starts, it is possible to check how the agents' predictions are faring throughout the day, using 15-minute delayed price data. A screenshot of this website is exhibited in Figure 87.

The iQuant website plays an important role in our research, because it enables us to publicly forward-test the multi-agent system. Ideally, every study on financial data mining should be

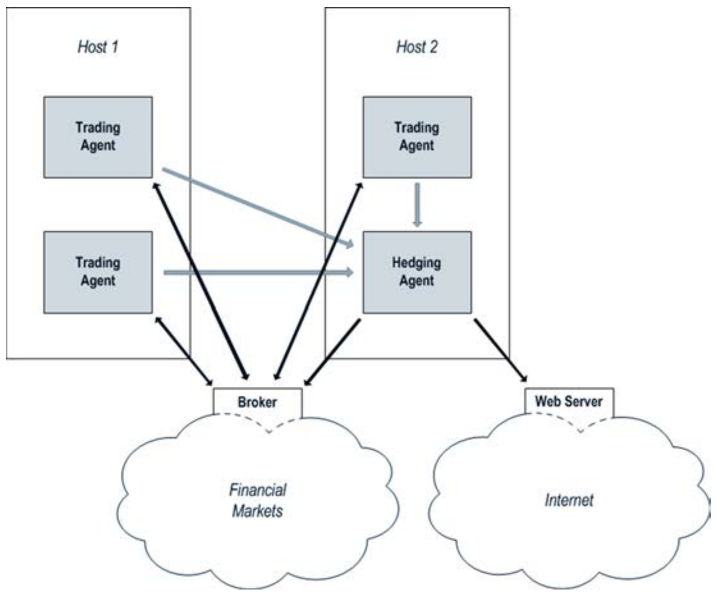
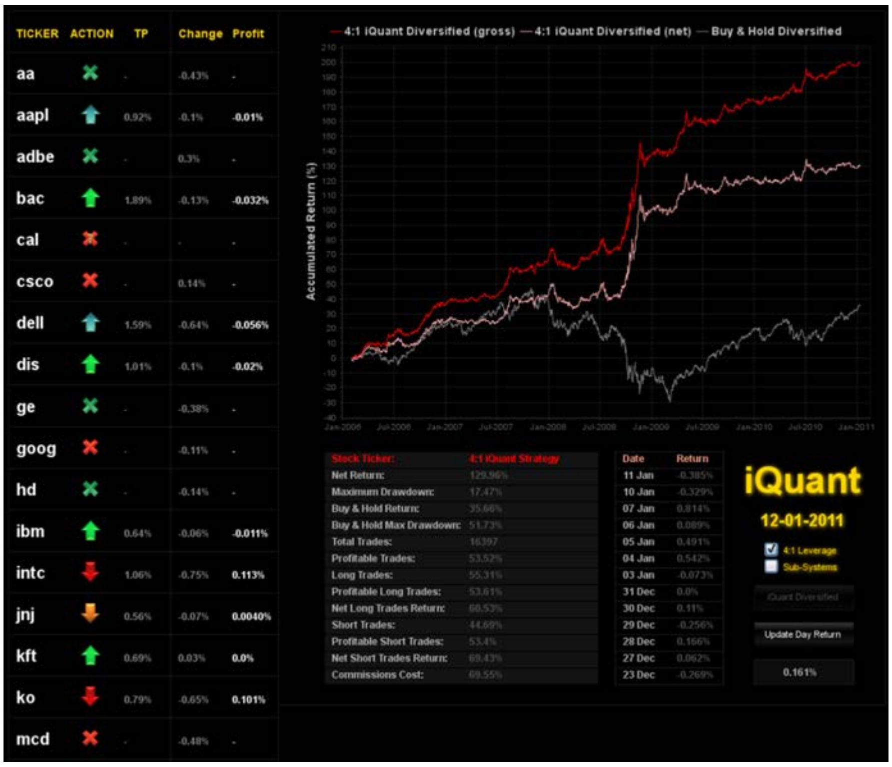


Figure 86. Graphical representation of a multi-agent stock trading system.

[12] The URL for the iQuant website is http://ruibarbosa.eu/iquant/iquant.html.

complemented with a similar resource. Publishing predictions in real-time is the only sure way of demonstrating that reported backtesting results are not biased; the importance of this demonstration should not be taken lightly, because biased conclusions are a recurring problem in studies on this subject (we stated our doubts regarding a few of these studies in previous chapters). Our multi-agent system has been publishing its predictions in the iQuant website since the beginning of 2009; so far, its performance has been in line with that of previous years. While there is no guarantee that it will continue to perform well in the future, this empirical evidence suggests that the system might have practical value. In the next chapter, we will discuss a couple of improvements that should make it even more valuable from the practical standpoint.



[✓] 4:1 Leverage
[ ] Sub-Systems

TICKER	ACTION	TP	Change	Profit
aa	✖	-	-0.43%	-
aapl	⬆	0.92%	-0.1%	-0.01%
adbe	✖	-	0.3%	-
bac	⬆	1.89%	-0.13%	-0.032%
cal	✖	-	-	-
csco	✖	-	0.14%	-
dell	⬆	1.59%	-0.64%	-0.056%
dis	⬆	1.01%	-0.1%	-0.02%
ge	✖	-	-0.38%	-
goog	✖	-	-0.11%	-
hd	✖	-	-0.14%	-
ibm	⬆	0.64%	-0.06%	-0.011%
intc	⬇	1.06%	-0.75%	0.113%
jnj	⬇	0.56%	-0.07%	0.0040%
kft	⬆	0.69%	0.03%	0.0%
ko	⬇	0.79%	-0.65%	0.101%
mcd	✖	-	-0.48%	-

Stock Ticker:	4:1 iQuant Strategy
Net Return:	129.96%
Maximum Drawdown:	17.47%
Buy & Hold Return:	35.66%
Buy & Hold Max Drawdown:	51.73%
Total Trades:	16397
Profitable Trades:	53.52%
Long Trades:	55.31%
Profitable Long Trades:	53.61%
Net Long Trades Return:	60.53%
Short Trades:	44.69%
Profitable Short Trades:	53.4%
Net Short Trades Return:	69.43%
Commissions Cost:	69.55%

Date	Return
11 Jan	-0.385%
10 Jan	-0.329%
07 Jan	0.814%
06 Jan	0.089%
05 Jan	0.491%
04 Jan	0.542%
03 Jan	-0.073%
31 Dec	0.0%
30 Dec	0.11%
29 Dec	-0.256%
28 Dec	0.166%
27 Dec	0.062%
23 Dec	-0.269%

Figure 87. Screenshot of the iQuant website.

Chapter 6

Intelligent Agents as Autonomous Index Traders

The trading agents described in previous chapters were all tested with just a few years' worth of out-of-sample data. Since our objective is to develop systems able to trade autonomously for an unlimited period of time, it is important to study their performance over a longer time span. That will be the main subject of this chapter. We will be describing the implementation of numerous index trading agents, which will be tested with data corresponding to a 25-year period. Most of these agents will be speculating on the daily value of the NASDAQ 100 Index, a stock market index that encompasses the 100 largest companies in the NASDAQ stock market. The reason why we chose this index is that its value has fluctuated wildly since inception, and hence its historical data is perfect for testing the adaptability skills of the trading agents. In addition to day trading, we will also configure the agents to trade with a weekly time frame, to determine if that will improve their accuracy. Finally, we will be experimenting with two different resource allocation strategies, with which we will try to increase the profit of the index trading agents. More specifically, we will be examining the effect of compounding and leveraging on their cumulative returns. We intend to demonstrate that both of these strategies are well suited for improving the performances of the multi-agent systems that were described in Chapters 4 and 5.

6.1 Trading Over Extended Periods of Time

The trading simulations that were previously described covered a relatively short period of time: the Forex agents were tested with just 2.3 years' worth of data, while the stock agents were initially tested with data equivalent to 3.3 years (this experiment is still ongoing, and may be followed online in the iQuant website). These experiments allowed us to analyse the performance of the trading agents in extreme market conditions. Overall, the results we got indicated that they were more or less successful at adapting to changes in price trends. Still, we believe it would be import to test their ability to adjust to much longer-term changes. The NASDAQ 100 Index is the perfect instrument to test that, because its value has been extremely volatile since its inception in 1985, as we can see in Figure 88. The speculative run-up between 1998 and 2000 is known as the dot-com bubble. This type of price movement – an exponential increase fuelled by greed, followed by a significant crash magnified by fear – is not uncommon in financial markets. Since no one really knows when exactly a price bubble will burst, it is difficult to trade profitably when these patterns occur. In order to test the resilience of our trading agent architecture in this scenario, we

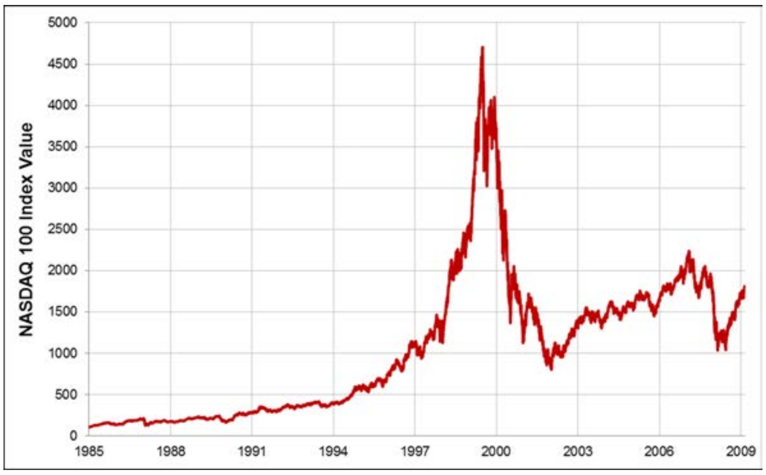


Figure 88. NASDAQ 100 Index since inception.

implemented an agent to day trade the NASDAQ_100 Index. We used the same procedure that was utilized with the stock trading agents, with the following differences:

- Instead of 11 models, we put 31 data mining models in the agent's ensemble, which were chosen by the automatic selection mechanism described in Section 4.3. To train the models, this mechanism used the "oldest" 50 instances, corresponding to the 2.5 months that followed the creation of the index in 1985; to test the models, the mechanism used the subsequent 50 instances. The sliding window for the test data was also set to 50 instances.
- The take-profit target was set to 7.5%; this target should not be hit very often, because daily price swings of this magnitude are rare.

The reason why we defined such a small training set was that we wanted to save as much data as possible for the trading simulation. Doing so allowed us to evaluate how well the agent would have performed, if it started trading the index more than two decades ago, with almost no initial knowledge. We were particularly interested in watching its behaviour at the index's inflection points. For example, in the year 2000, its data mining models should be more inclined to predict price increases than price decreases, because most training instances up to that point covered periods in which the price trended upward very vigorously. The agent's success will be determined by how quickly it adapts to the dramatic crash that occurred in the following years. Figure 89 shows its cumulative return throughout the full simulation period. We can verify that this agent was able to pick up on the index's initial uptrend very quickly; more importantly, the stock market crash in the year 2000 did not have a very significant impact on its return, which implies its prediction module adjusted to the new market conditions relatively fast, before any major losses could occur. We must note that this return curve is not very realistic, because it does not include the trading costs. It is hard to define a cost per trade to use in this simulation, because index trading has changed a lot since 1985. Nowadays, one can trade indices using a multitude of instruments,

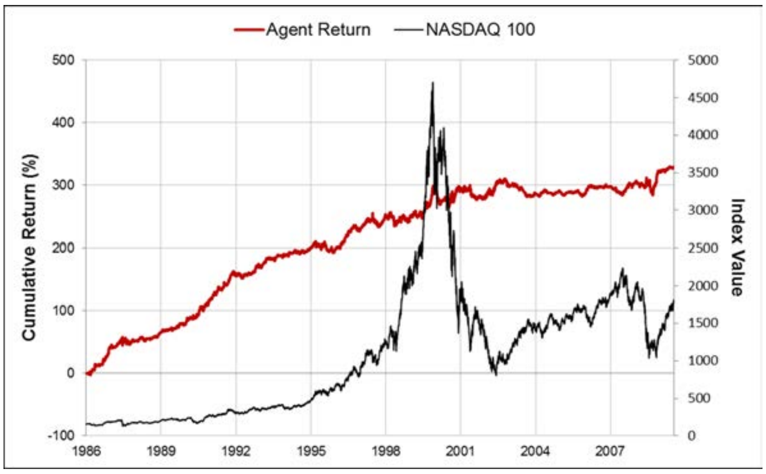


Figure 89. Gross cumulative return of the NASDAQ 100 trading agent, compared with the index's value throughout the simulation period.

among which exchange-traded funds, futures and contracts for a difference, all of which are relatively cheap. Most of these instruments did not even exist in the 1980s, or were much more expensive to trade at the time, so any trading simulation going that far back is bound to be unrealistic from a practical point of view. But the purpose of this experiment is to test the adaptability skills of the agent, not to obtain lifelike results. In that regard, we can conclude that the agent performed quite well, considering the challenging conditions it faced. It made good trading decisions when the index was trending upward, and also when it was trending downward, which is exactly the kind of behaviour we were hoping it would exhibit. To put this performance into perspective, we compared it with that of four simple trading bots, each of which utilized one of the following naïve strategies:

- always buy, i.e., buy every day at the beginning of the session, and sell at the end;
- always sell, i.e., short sell at the beginning, cover at the end;
- buy if the index increased in the previous day, short sell otherwise;
- buy if the index decreased in the previous day, short sell otherwise;

The comparison is shown in Figure 90 and in Table 21. Clearly, the agent's strategy was much better than the simpler ones. It was more accurate, opened less trades, and achieved a bigger profit with smaller risk. The results of this simulation demonstrate why it is dangerous to extrapolate from past performance to future returns. Consider the cumulative return of the bot that bought the index when its value went up in the previous day, and shorted it when it dropped. This strategy worked perfectly throughout the technology bull market, and so the bot did great up until 1999. However, this same strategy has been disastrous ever since – it became useless as soon as the market started collapsing. Successful long-term trading requires the ability to adapt to these changes; in that respect, the greater complexity of the agent's strategy clearly paid off.

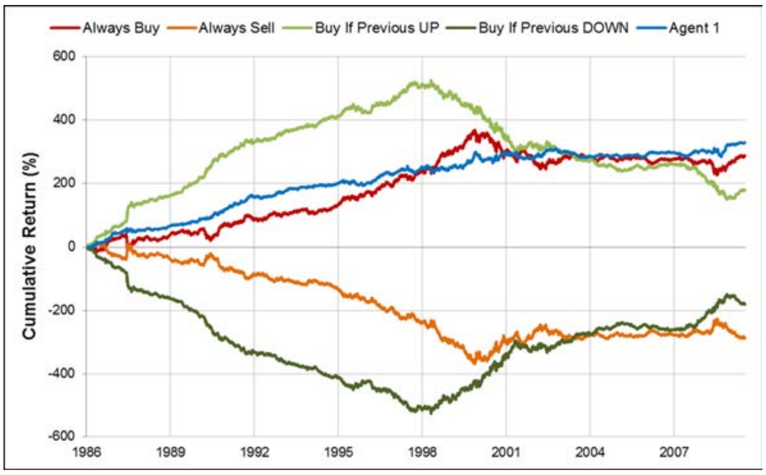


Figure 90. Gross cumulative return of the NASDAQ 100 trading agent, compared with four naïve trading bots.

Table 21. Simulation results of the NASDAQ 100 trading agent, compared with four naïve trading bots (excluding trading costs).

Strategy	*Ret (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Acc (%)*	*Succ (%)*	*Trades*
Always Buy	287.1	142.8	2.01	0.0483	53.4	53.4	5,939
Always Sell	-287.1	383.5	-0.75	-0.0483	46.6	46.6	5,939
Buy If Previous UP	180.7	377.9	0.48	0.0304	52.4	52.4	5,939
Buy If Previous DOWN	-180.7	527.5	-0.34	-0.0304	47.5	47.5	5,939
Agent 1	329.3	35.1	9.39	0.0956	54.7	54.7	3,443

Since the data mining models in its ensemble were selected randomly, it is possible that the agent's ability to "survive" the bursting of the dot-com bubble was just a fluke. In order to verify if other agents would exhibit the same resilience, we implemented four new ones, using the exact same method. Their simulation results are displayed in Figure 91 and in Table 22, together with the results of the first agent that was tested (agent 1). Not surprisingly, we can see that the choice of models did indeed have a major impact on the performance. Compared with agent 1, agent 5 performed better (higher RMD and higher return per trade), while agent 3 performed much worse. Disappointingly, agents 2, 3 and 4 all experienced significant losses when the NASDAQ 100 Index crashed. Still, those losses were small compared to their gains up to that point, and it looks like they were on their way to recovering from the drawdowns at the end of the simulation period, albeit very slowly. In view of these results, we may conclude that the agents built according to the proposed architecture will only be as good as the strategy employed to select their data mining models. Our random selection method allowed us to implement some profitable agents, but is nonetheless very limiting, in that it does not let us optimize them in any way. A sounder method would likely generate better agents.

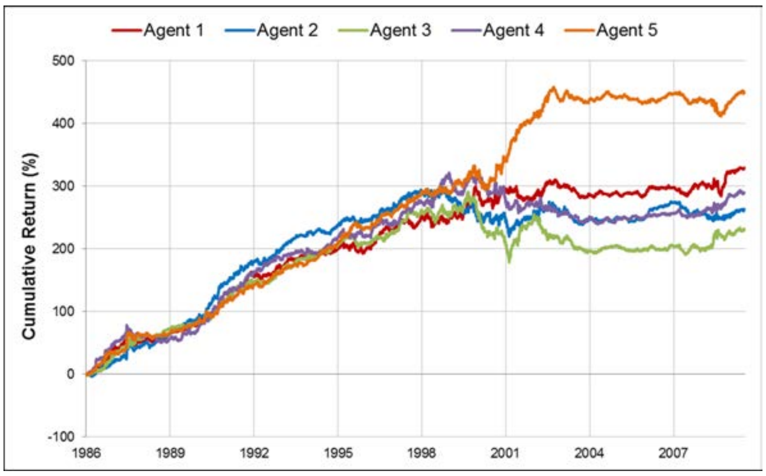


Figure 91. Gross cumulative returns of five different NASDAQ 100 trading agents.

Table 22. Simulation results of five different NASDAQ 100 trading agents (excluding trading costs).

Agent	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Success (%)*	*Trades*
1	329.3	35.1	9.39	0.0956	54.7	54.7	3,443
2	262.5	78.2	3.36	0.0756	54.4	54.4	3,472
3	231.1	112.3	2.06	0.0639	54.1	54.1	3,615
4	289.6	81.7	3.54	0.0820	53.4	53.4	3,531
5	450.4	46.4	9.71	0.1309	54.5	54.5	3,441

6.2 Trading With Longer Time Frames

We have previously suggested that increasing the investment time frame might improve the accuracy of the trading agents, because the price data will be less noisy. To test this assumption, we trained five agents to trade the NASDAQ 100 Index with a weekly time frame – they opened each trade on Monday, at the beginning of the trading session, and closed it on Friday, at the end of session (or during the week, if the take-profit rule was activated). We used the exact same procedure with which the day trading agents were implemented, only the test data sliding window was set to 8 instances, which encompass around 2 months' worth of price data. The trading simulation results of the five agents are show in Figure 92 and in Table 23. As expected, the average accuracy of the agents with the longer time frame was higher than that of the day trading agents – 54.4% versus 54.2% – but this increase was not statistically significant. Despite the slight improvement, the agents' precision was still far from impressive. We should point out that, besides the bigger time frame, there are other changes that we could make to try to improve their accuracy, such as defining better training attributes, or a better method for selecting the models. The latter is particularly important because, once again, we can verify that different ensembles trained with the same data achieved completely different results.

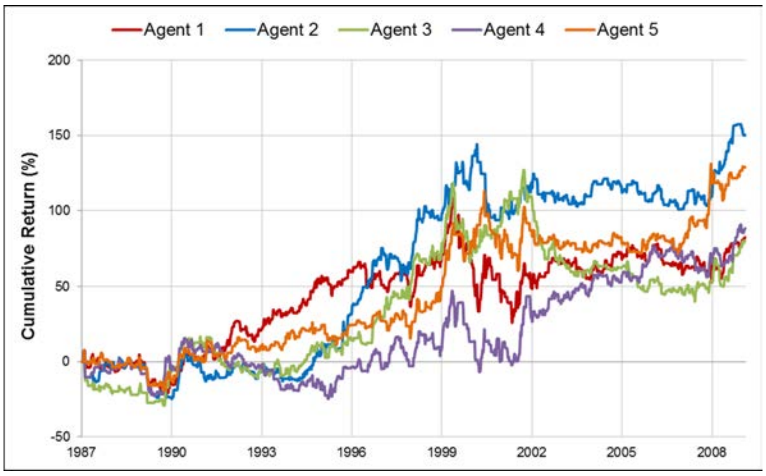


Figure 92. Gross cumulative returns of five different NASDAQ 100 agents with a weekly trading time frame.

Table 23. Simulation results of five different NASDAQ 100 agents with a weekly trading time frame (excluding trading costs).

Agent	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Success (%)*	*Trades*
1	82.1	82.4	1.00	0.1289	55.7	55.7	637
2	150.4	50.2	3.00	0.2507	54.8	55.3	600
3	80.1	86.9	0.92	0.1314	52.3	52.8	610
4	88.2	53.8	1.64	0.1341	54.1	54.4	658
5	128.8	52.3	2.46	0.2241	55.0	55.5	575

Since all the return curves show a distinct upward trend, it may be said that all the agents accomplished their purpose. However, their performance as a whole was not very good. They all had very large drawdowns, with comparatively small returns, which would be even smaller if we subtracted the trading costs. The biggest losses occurred in 2000, when the index's trend suddenly changed. Given the magnitude of this change, and the speed with which it occurred, it is easy to see why the longer time frame did not work in the agents' favour. As we already know, the trading agents require some time to adapt to new market conditions, because their models need to be properly reweighted to reflect those conditions; this becomes a problem when the time frame is longer, because the agents are exposed to bigger price swings that might cause massive losses. In the year 2000, for example, the value of the NASDAQ 100 Index dropped dramatically from one week to the next, so the agents suffered several big losses in a row, before they could adjust to the new trend. One possible solution to this problem would be to define a stop-loss rule in their domain knowledge modules, which would cap the maximum loss per trade. This is something that should be considered for any agent investing real funds. We cannot, however, simulate the use of a stop-loss rule when the agents are trading with a weekly time frame, because the results of that simulation would be very deceiving: since an instrument's price may gap up or down in-between trading days, there is no way to tell, in retrospect, if and at what price the stop-loss rule would be activated.

In order to see if the weekly time frame was better suited for trading other indices, we created two new agents: one to trade the S&P 500, and the other to trade the Dow Jones 30. Their simulation results are shown in Figures 93 and 94, and summarized in Table 24. Both agents performed poorly; their returns were miniscule compared to the rise in value of the two indices, from beginning to end: the S&P 500 increased 340%, while the Dow Jones 30 increased 427%. The worst thing about their performances is that they both traded profitably up to a certain point, and then entered a drawdown from which they could never recover, even after many years of

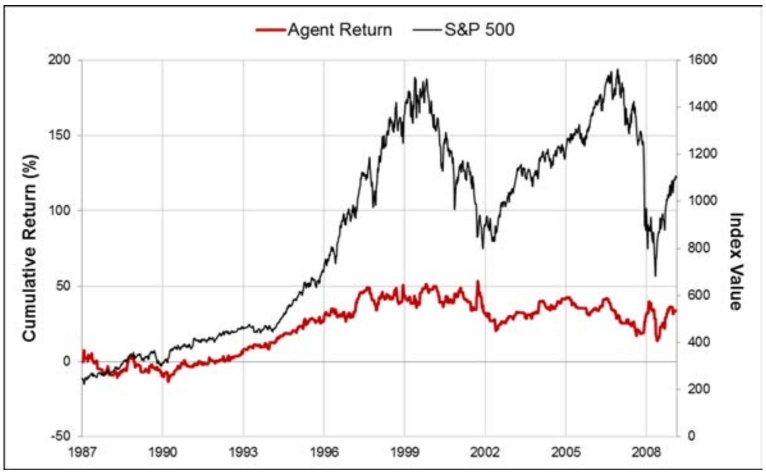


Figure 93. Gross cumulative return of the S&P 500 agent with a weekly trading time frame.

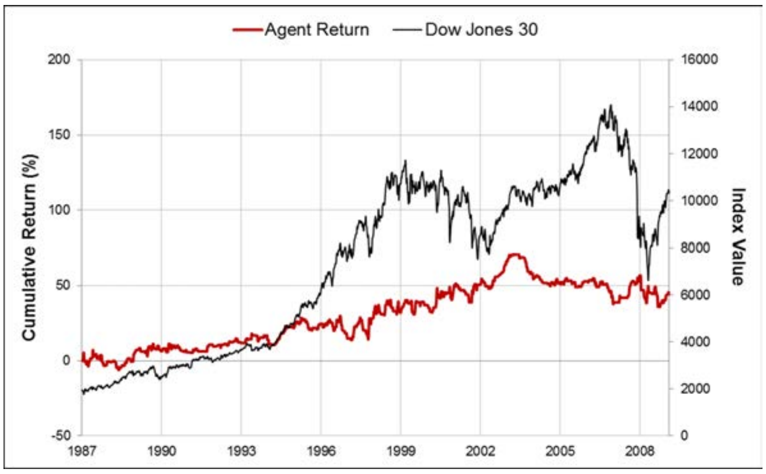


Figure 94. Gross cumulative return of the Dow Jones 30 agent with a weekly trading time frame.

Table 24. Simulation results of index trading agents using a weekly time frame (excluding trading costs).

Agent	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Success (%)*	*Trades*
S&P 500	33.9	39.4	0.86	0.0520	52.5	52.7	651
Dow Jones 30	44.4	35.0	1.27	0.0794	53.0	53.0	559

trading. This implies that our agents may become unsuccessful at any time. Therefore, we need to be open to the idea that, in a production system, some agents might eventually need to be "fired", and replaced with new agents. This is not much different from what happens in a traditional investment company, where traders face the same fate when they do not perform well.

Overall, the results presented in this and in the previous section indicate that the daily time frame is the better option for the index trading agents: it gives them more time to adapt to trend changes, before any substantial losses can occur, and also provides them with a lot more profit opportunities (although in real life this would imply more trading expenses). Despite the disappointing results, longer time frames should not be disregarded – they might be better fitted for less volatile instruments, and will definitely be useful when developing diversified systems, because mixing agents with different time frames will improve the investment diversification.

6.3 Compounding for Better Returns

In the trading simulations described up to this point, all the agents employed the same resource allocation strategy:

- the standard trade size was set equal to the initial trading capital (or part of it, in diversified systems), and the agent's return and maximum drawdown were calculated as a percentage of that capital;
- regardless of its accumulated losses and gains, the amount that the agent invested in each trade was always the initial, or half the initial trading capital (i.e., the standard or half the standard trade size).

Since the amounts invested are fixed (regardless of the losses) the agent can lose more money than the initial capital; put another way, its maximum drawdown may be over 100%. When the cumulative return falls below zero – meaning the agent has less money than what it started with – it is possible that the size of its trades will be bigger than the capital available, and so these trades will require borrowed funds (i.e., leverage). We believe that configuring the agent to vary the size of the trades according to the money it possesses should yield better returns. In order to do this, the agent must change the standard trade size parameter before each trading period, setting it to the money available at that point. In doing so, the amount invested will increase when the agent is trading successfully (meaning the gains are reinvested, or compounded), and will decrease when it suffers losses, which prevents it from losing more money than what it is given initially. Figures 95 and 96 display the cumulative returns obtained by the NASDAQ_100 day trading agents 1 and 3, when using this resource allocation strategy, in comparison with the returns they got when using a fixed standard trade size. The full simulation results are summarized in Tables 25 and 26, presented in the next section. We draw two conclusions from these results. First of all, it is clear that compounding can greatly improve the profitability of the trading agents – Figure 95 shows that agent 1 was able to achieve an extraordinaire return with this strategy. Secondly, this experiment demonstrates that compounding will only be useful if the agent's investment strategy is not prone to big drawdowns. It is easy to see why: since the trade size increases as the agent accumulates more profit, more money will be put at risk in each consecutive trade, which means large drawdowns can happen really fast, and cause very big losses. An example of this drawback is shown in Figure 96: up to the year 2000, agent 3 was making a lot of profit by continuously reinvesting its gains; however, when the tech bubble burst, it lost most of that profit due to a series of big-sized unsuccessful trades.

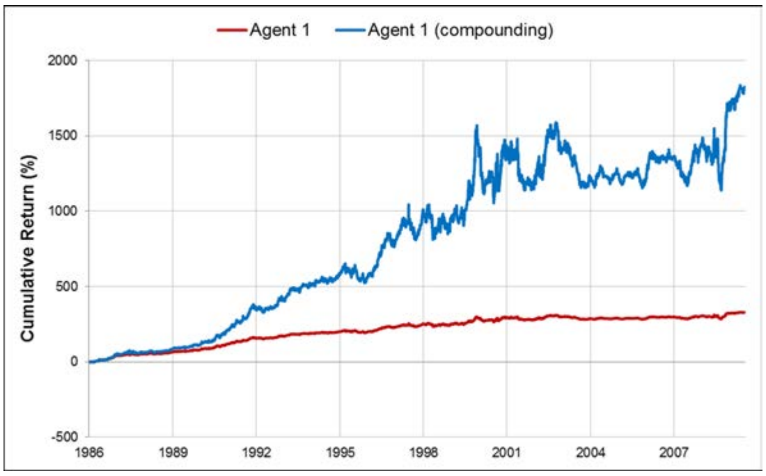


Figure 95. Gross cumulative return of the NASDAQ 100 trading agent 1, with and without compounding.

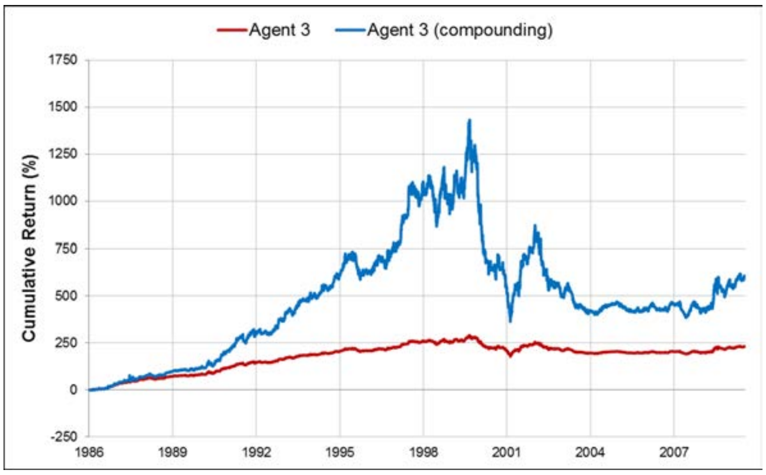


Figure 96. Gross cumulative return of the NASDAQ 100 trading agent 3, with and without compounding.

As mentioned before, an important advantage of the compounding strategy is that the agents will never lose more money than what we give them, so this is a good way to limit the risk. For instance, agent 3 would have suffered a maximum loss of 112.3% if configured to trade with a fixed standard trade size (according to Table 22). Had it begun trading exactly when that drawdown occurred, it would have lost all the money, and possibly more if the broker did not forcefully close its positions. On the other hand, with compounding, the maximum drawdown that a day trading agent may experience is capped at 100%, because it will never invest more money than it has. This explains why agent 3's maximum loss dropped from 112.3% when using a fixed trade size (measured as the biggest peak-to-valley drop in its cumulative return curve) to 69.8% when using a variable trade size (also measured as the biggest peak-to-valley drop in the cumulative return curve, but calculated as a percentage of the capital at the peak). We should note that, even when compounding, the agent can still lose more that 100% of the money, in a very specific scenario: if it short sells a financial instrument, and then the price soars to more than double during the day, at which point the broker will close the position. Trading halts may also originate similar problems, if the agent gets stuck with an open trade. Nevertheless, both situations are very rare.

The results shown in this section should clarify why we put so much emphasis on risk management when designing the trading agent architecture. If an agent is able to make small gains consistently, without experiencing any major losses in-between, it will achieve extraordinary returns by simply compounding those gains. Thus, to create a successful trading system, we do not need it to have a very high average return per trade; the only requirement is that it is capable of accumulating small gains, without suffering any big drawdowns. Interestingly, the diversified multi-agent systems that were presented in Chapters 4 and 5 fit this description very well.

6.4 Leveraging for Better Returns

Compounding enables us to improve the agents' returns without increasing the risk. Trading with borrowed funds, on the other hand, allows us to multiply the returns, but at the expense of proportionally bigger drawdowns. This is yet another reason why we focused so much on risk management. Since leverage will greatly increase the trading risk, the agents should only be permitted to use borrowed funds if their investment strategies are relatively safe to begin with, otherwise they will inevitably end up getting a margin call. Figures 97 and 98, and Tables 25 and 26, show the improvement to the performances of the NASDAQ 100 day trading agents 1 and 3, when configured to employ a maximum initial leverage of 4:1. To make them use this leverage, we just have to quadruple their standard trade size (i.e., set it to four times the initial capital); by doing so, their investment in each trading period may be up to four times the starting balance, and hence they can lose more money than what they started with (although that should be prevented by the broker's margin call).

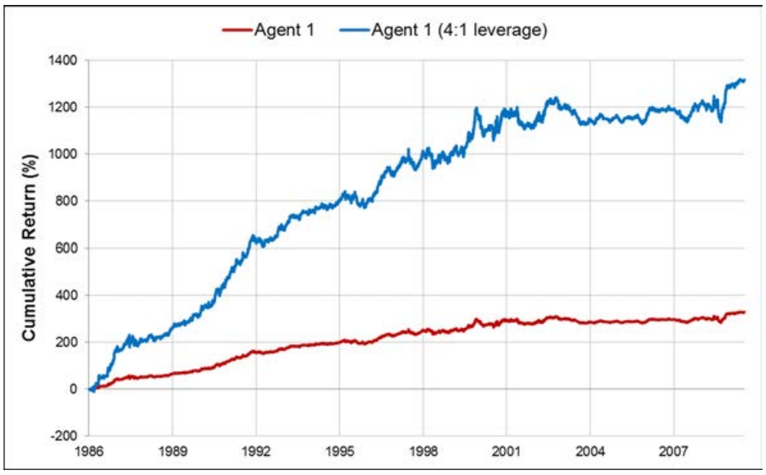


Figure 97. Gross cumulative return of the NASDAQ 100 trading agent 1, with an initial maximum leverage of 1:1 or 4:1.

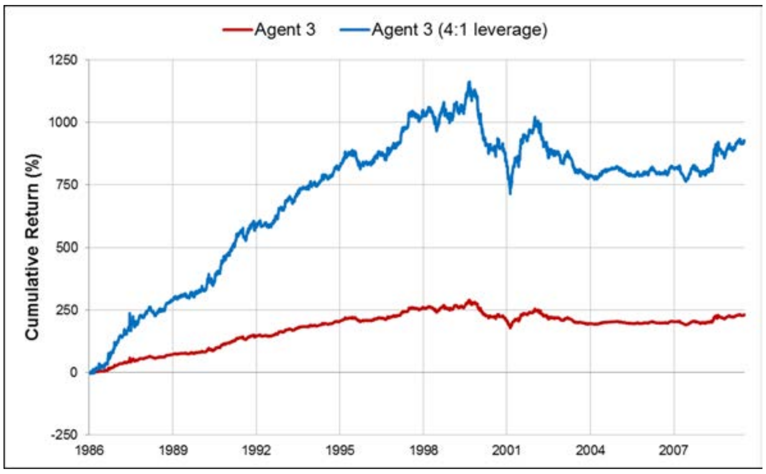


Figure 98. Gross cumulative return of the NASDAQ 100 trading agent 3, with an initial maximum leverage of 1:1 or 4:1.

Tables 25 and 26 reveal that, as expected, the use of 4:1 leverage quadrupled the agents' returns, as well as their maximum drawdowns: agent 1 had a profit of 1,317.2% and a maximum accumulated loss of 140.2%, while agent 3 had a profit of 924.4% and a maximum loss of 449.3%. If these were live trading results, the enormous drawdowns would not be very problematic, because they occurred at a time when both agents had already accumulated a lot of profit. However, if they started trading right when those losses occurred, they would have lost all their money in just a few days, and would not be able to continue trading. Hence, even if they did yield amazing returns during the simulation period, it is clear that the index trading agents – or any other standalone agents, for that matter – should not be allowed to trade real funds with 4:1 leverage, because the potential risk is unacceptable. Such high leverage should be reserved for trading strategies that have proven to be extremely safe in the past, like those of the diversified multi-agent systems that we described previously. In the next chapter, we will present an even more diversified strategy that should be well suited for using both leverage and compounding.

Table 25. Simulation results of the NASDAQ 100 trading agent 1 using different resource allocation strategies (excluding trading costs).

Strategy	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Success (%)*	*Trades*
Standard	329.3	35.1	9.39	0.0956	54.7	54.7	3,443
Compounding	1,824.7	30.9	59.09	0.5300	54.7	54.7	3,443
4:1 Leverage	1,317.2	140.2	9.39	0.3826	54.7	54.7	3,443

Table 26. Simulation results of the NASDAQ 100 trading agent 3 using different resource allocation strategies (excluding trading costs).

Strategy	*Return (%)*	*Max DD (%)*	*RMD Ratio*	*Ret/Trade (%)*	*Accuracy (%)*	*Success (%)*	*Trades*
Standard	231.1	112.3	2.06	0.0639	54.1	54.1	3,615
Compounding	601.0	69.8	8.61	0.1662	54.1	54.1	3,615
4:1 Leverage	924.4	449.3	2.06	0.2557	54.1	54.1	3,615

Chapter 7

The Autonomous Multi-Agent Hedge Fund

Our intelligent trading agents were designed to be, more or less, the software equivalent of human traders. Hence, if we put enough of them together, and integrate them into a single system, we should end up with the software equivalent of a hedge fund. In this AI-based fund, the agents would be responsible for all the investment decisions, while human intervention would be relegated to basic management tasks, like "firing" the least capable agents, and configuring new ones. This concept represents the culmination of our efforts to replace human traders with intelligent agents. Since the agents we created seemed skilful enough to emulate the actions of successful traders, the next logical step should be to incorporate them into a multi-agent system, and build an investment company around it. In this chapter, we will explain the reasons why the creation of one such company would be a good idea, by listing its many advantages over traditional trading ventures. Afterwards, we will describe an example of a multi-agent system that could be utilized for that purpose. This system is composed of the numerous trading agents that were introduced in Chapters 4 and 5; it contains sufficient agents and is diversified enough that we could use it to create a small autonomous "intelligent" hedge fund. We will test the system's trading proficiency with 2.3 years' worth of out-of-sample data, and will compare its results with the industry's average.

7.1 Motivations for an Agent-Based Hedge Fund

The main objective that was initially established for our research was the investigation of the potential use of intelligent agents as autonomous financial traders. For that, we designed an agent architecture from the ground up, hoping to define a framework that would allow us to create talented trading agents. Looking at the results presented in Chapters 4, 5 and 6, we believe that objective was achieved. Many of the agents described in those chapters were able to trade profitably with low risk, particularly when inserted in small diversified multi-agent trading systems. As is, these agents and systems could prove very useful for traditional companies in the financial field – for example, they could be utilized to aid human traders, or to complement pre-existing investment strategies. But in our opinion, they hold the potential to play an even bigger role in the industry. Given the way our work progressed, we can envision an entire investment company built solely around software agents: an autonomous "intelligent" hedge fund. A regular hedge fund is basically just an investment company in which traders attempt to obtain above average returns, by buying and short selling different types of financial instruments (stocks, derivatives, currencies, bonds, etc.) using leverage. Implementing a multi-agent system with the same modus operandi would be straightforward. This system would have significant advantages compared to traditional companies. First of all, as we have previously stated, intelligent trading agents offer many advantages over human traders: they do not receive salaries or bonus, get no vacation days, and can work 24 hours a day uninterrupted. Also, given that most investment analysis (be it technical or fundamental) is nothing more than number crunching and pattern matching, it is fair to say that intelligent agents should be better suited for this task than human traders. If there is indeed a scientific explanation behind the success of profitable human traders – i.e., their success is not due to fraud (insider trading, misrepresenting results, market cornering) or to luck – then there is no reason why their strategies cannot be taught to intelligent agents, which would then be able to trade more reliably, for a fraction of the price. Finally, we must also point out that the actual performance of regular hedge funds is far from outstanding. In Section 1.3, we looked at the Barclay Hedge Fund Index, and concluded that the average hedge fund should not be capable of offering above average returns in the long run. This conclusion is even clearer when we focus on the returns of the hedge funds that specialize in investing on both the long and the short sides of the market simultaneously, without being market neutral (which is basically the strategy employed by our trading agents). The performance of these funds is viewable in the Barclay Equity Long/Short Index[13], an average of the returns of several hundreds of funds that follow that strategy. The returns throughout the last decade are displayed in Figure 99. For the period between 2000 and 2009, the hedge funds achieved an average annual return of 7.2%, while Treasuries yielded 4.5%, and the S&P 500 returned 1.2%. Notice these are the returns that we would get by investing the same amount at the beginning of each year. If one were to buy at the beginning of 2000, and kept reinvesting the money returned at the end of each year (i.e., compound the investment), the average annual returns

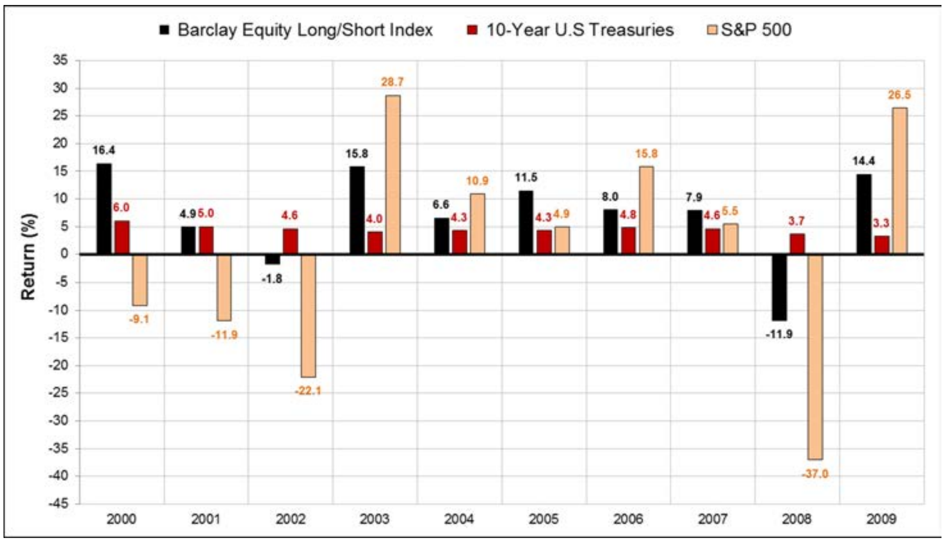


Figure 99. Net annual return of the Barclay Equity Long/Short Index, the 10-Year U.S. Treasuries and the S&P 500 Index.

[13] Available at http://www.barclayhedge.com/research/indices/ghs/Equity_Long_Short_Index.html.

would be significantly different: 9.4% for the hedge funds, 5.5% for the Treasuries and 0% for the S&P 500. Not surprisingly, Treasuries were the safest investment of all. As for the profit, the hedge funds' was not great, but they did at least provide the best return compared to the other two strategies. However, as we saw in Section 1.3, it is extremely likely that the values of the index are very inflated, due to all sorts of biases (e.g., the worst hedge funds that drop out of the index often do not report their losses). Ergo, it is doubtful that the average hedge fund can really outperform passive investment strategies. This is clearly not a very efficient industry, and so it should not be that difficult for a real life agent-based hedge fund to stand out. The fact that this system would have many advantages over traditional hedge funds (lower costs, no rogue traders, more reliability), in addition to the fact that the average hedge fund carries a lot of risk, and is not even that profitable, are the reasons why we believe the creation of an agent-based hedge fund is a worthwhile idea, at least in theory. In the next section, we will try to prove that this idea also makes sense in practice, by testing one of these systems in lifelike conditions.

7.2 Performance Analysis

Each of our intelligent agents emulates a human trader specialized in the trading of a specific financial instrument. By grouping many of these agents together – to diversify the investments and lower the risk – we can implement an autonomous trading system that will be able to interact in multiple markets without human supervision. This system holds the potential to be utilized as an autonomous "intelligent" hedge fund, i.e., a hedge fund where the intelligent agents are charged with making all the trading decisions, and the humans are merely system administrators. In order to create an entity that fitted this description, we merged our multi-agent currency trading system, described in Chapter 4, with our multi-agent stock trading system, described in Chapter 5. To do this, we just had to ensure that all the agents shared the same resources, meaning all their profits and losses ended up in the same trading account. Creating an investment company around this bigger and more diversified multi-agent system would be an interesting proposition. This company would consist of 10 currencies traders (with a 6-hour investment time frame), and 25 stock traders (with a daily time frame). We tested the system by having it simulate trades throughout the period between February of 2007 and May of 2009. Its trading results are shown in Figure 100 and Table 27. As expected, diversifying the investment with different types of financial instruments led to a substantial decrease in the trading risk: the maximum drawdown of the hedge fund system was an almost negligibly 2.9%, which is much less than either of its constituent multi-agent systems experienced on their own. As for the return, the hedge fund system yielded a profit of 20.7% at the end of the simulation period, corresponding to a gain of 4.1% in the last 11 months of 2007, 13.5% in 2008 and 3.1% in the first 5 months of 2009. Despite the low risk, this performance is not that great. Based on our own trading experience, we consider a good return to be somewhere between 10% and 15% per annum in a low interest rate environment – a lower return would not justify the risk (versus safer investments like government bonds), while a higher return might imply too much risk (and would likely not be sustainable in the long run). Fortunately, there are ways to improve the profit of the multi-agent system, so that it matches our expectations. One characteristic that distinguishes hedge funds from other types of investment companies is that they are allowed to use

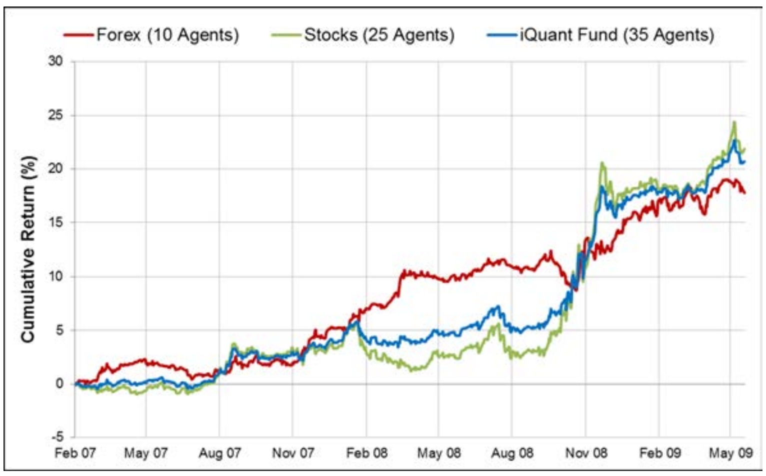


Figure 100. Net cumulative return of the intelligent hedge fund, compared with the individual multi-agent systems it combines.

Table 27. Simulation results of the intelligent hedge fund from February of 2007 till May of 2009, compared with the individual multi-agent systems (including trading costs).

System	*Return (%)*	*Max DD (%)*	*RMD Ratio*
Forex Trading	17.8	3.8	4.72
Stock Trading	21.9	4.4	5.00
Hedge Fund	20.7	2.9	7.18

leverage and to compound their returns. In the previous chapter, we saw that both of these strategies can dramatically improve the performance of a trading agent, so long as its investment strategy is not very risky to begin with. Since our hedge fund system proved to be extremely safe throughout the simulation period, it would be appropriate to let its agents compound the gains and trade with borrowed funds; this should increase the return of the system as a whole, without (hopefully) giving way to any dangerous drawdowns. Figure 101 shows the cumulative return obtained by the system in the simulation period, when the agents are permitted to use a maximum initial leverage of 4:1, and are configured to use variable trade sizes (to reinvest the gains). In this scenario, the system's maximum drawdown, measured on a daily basis, worsened to 11.2%, but its final return improved to a staggering 119.7%, or 17.1% in the 2007 period, 77.8% in 2008, and 24.8% in the first 5 months of 2009. This would be, by any standards, an outstanding performance.

If we were to compare the statistics of the leveraged multi-agent system with those of real hedge funds in the same category, our system would definitely rank among the best. However, this comparison would be disingenuous. First of all, 2.3 years of simulated trading is far too short of a period to draw any definite conclusions regarding the system's viability in the long run. On the bright side, this period already includes a tail event (the subprime crisis), which the system withstood without any problems. Still, we would need to test it over a much longer time span, in order to be able to conclude that it can really overcome any dramatic changes in market conditions. Also, we cannot compare our simulation results with those of real life traders, because the

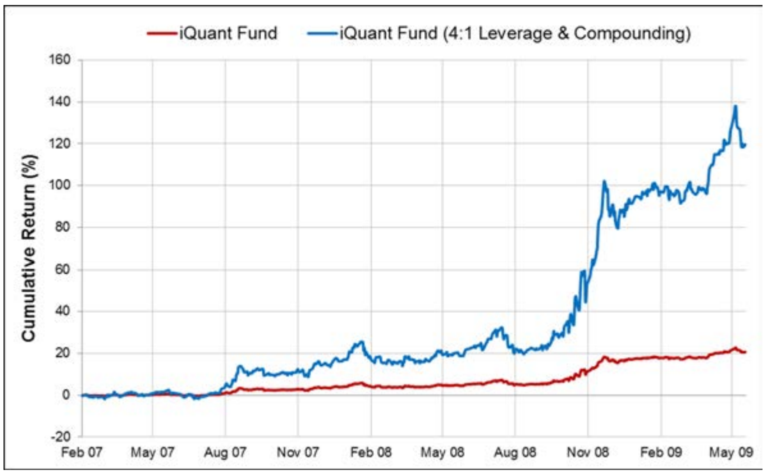


Figure 101. Net cumulative return of the hedge fund system with compounding and an initial maximum leverage of 4:1.

simulation environment does not account for numerous problems that may occur while trading live, among which:

- What if the price moves too fast, and the agents cannot open or close trades at the desired price?
- What if there is a connection problem with the broker, and the agents are unable to open or close trades?
- What if the market rules change? This happened in 2008, when the U.S. Securities and Exchange Commission imposed a temporary ban on the short selling of hundreds of securities, among which the BAC and the GE stocks.
- What if the broker goes bankrupt? Unlikely, but not unheard of.

Any one of these issues could compromise the real life performance of the multi-agent system. It is possible that, on some occasions, some of these problems might actually improve its return. For example, if an agent buys a stock, and is later prevented from selling it due to a network disconnection, this could turn out to be beneficial if the price of the stock keeps going up. However, we are more inclined to believe that Murphy's Law would apply in these situations. The implication here is that any simulated trading results must be taken with a grain of salt. While they offer some insight regarding how well a system might perform when trading with real funds, they do not provide any guarantees. This uncertainty is characteristic of the investment field, and there is simply no way around it. Even if we had an extensive live-trading track record for the agent-based hedge fund, we would still not be able to make any assumptions regarding its performance going forward, because past performance does not guarantee future returns. Despite these shortcomings, we believe our experiment showed some very encouraging results. By combining the 35 agents in a single multi-agent system, we implemented a relatively safe investment strategy that not only survived the most extraordinary market conditions, but actually thrived with the greater volatility. Even if its simulation results present some limitations – they do not account for all the intricacies of real life trading, and cannot be used to extrapolate the future performance of the system – they were at least good enough to warrant giving the intelligent agents the chance to prove their worth in the real markets. Several of these agents were very skilful throughout the simulation period, which leads us to believe that the practical usefulness of their architecture (described in Chapter 3) will be vindicated once they are allowed to trade real funds.

Obviously, there are still a lot of improvements that could be made to the proposed agent-based hedge fund, like adding more agents to it, defining better methods to train its agents, or even coming up with better agent architectures. It is fair to say that our experiments have barely scratched the surface of what can be accomplished with this type of system. So, we can conclude that our research has revealed, at the very least, the enormous potential that exists for the application of artificial intelligence and agent technology in the world of finance.

Chapter 8

Conclusions and Future Work

For better or worse, the financial industry impacts the life of every person in the civilized world. Energy and food prices, currency exchange rates, government and corporate bond yields, mortgage rates, as well as many other variables, are all directly affected by the industry's players. This influence is exerted through the participation in financial markets, of which there are various types. Participants in commodity futures markets, for example, set the prices of goods such as cotton, corn, live cattle, oil, natural gas, gold, and even frozen concentrated orange juice and pork bellies, among numerous other products. Bond markets, on the other hand, serve the purpose of facilitating the trading of government and corporate debt. Besides issuing debt, companies can also raise capital in the stock markets, by selling shares to entities interested in owning a stake in them. One other market that has a major influence on our everyday lives is the foreign exchange, where currency prices are set. A huge services industry has flourished around these financial markets, with commercial and investment banks, insurance companies, mutual funds, hedge funds, brokers, market makers and retail investors all trying to beat the averages, using ever more sophisticated strategies. Much of the activity occurring in these markets boils down to speculative trading (i.e., the traders attempt to anticipate price movements, and make risky transactions based on inconclusive evidence). This activity does not get much respect from industry outsiders, because speculators are often seen as price manipulators; nevertheless, they have an important role as liquidity providers (meaning, there are always traders willing to buy or sell the financial products at all times), which is essential to keep the markets running smoothly.

Thanks to advances in technology, it is now easier than ever to trade in these markets – we can place an order in an online broker, and have it forwarded to an exchange on the other side of the world in a fraction of a second. In addition to facilitating the interactions between the traders, technology is also becoming an important part of the decision process. According to various reports in the media, an increasing number of trading companies are putting computers in charge of analysing the financial data, and opening the trades. However, except for a few buzzwords, not much is known about their systems, or the alpha they produce (i.e., the excess return they offer over simpler benchmark strategies with similar risk). It is easy to understand why these companies are turning their attention to technology: computers excel at parsing huge amounts of data and finding hidden patterns in it, and are able to place trades much faster than human traders, hence they can be useful in every step of the trading process. The premise for our research was that, rather than aiding human traders, computers might actually be capable of replacing them altogether. Assuming that successful traders owe their success to sound reasoning and methodical behaviour, it should be possible to teach their investment strategies to software agents. However, this is an arguable assumption. The most cynical critics of speculation maintain that successful traders are more lucky than skilful – given the many thousands of participants in financial markets, if the returns over their lifetime follow a normal distribution, then some (the outliers) will inevitably emerge as big winners, simply because they were fortunate enough to end up on the right side of the bell curve. Those who believe in rational and efficient markets would also be quick to point out that financial prices always fully reflect all the information available, so no one should be able to beat the markets (i.e., to consistently obtain above average returns) unless it happens by chance. The way we see it, the numerous bubbles and crashes in asset prices that occurred in the last decade have already proven, beyond any doubt, that financial markets are far from rational. During the downturns, many funds have been forced to liquidate their holdings, not because they thought that was the right thing to do at the time, but because they got margin calls or were flooded with redemption requests from panicking customers; this selling was not a rational decision – the funds had to sell their assets at any price, even if they believed those assets were worth more. Conversely, in the upturns, it is common for the "irrational exuberance" to take over the markets, as greedy investors pile their bets in ever growing asset bubbles. Since market participants do not always act rationally, it is entirely possible that some asset prices could be at odds with their intrinsic value from time to time; hence, there will be profit opportunities that may be exploited by those talented enough to spot them. This belief is at the basis of the trillion-dollar financial services industry. While we cannot completely disregard the idea that chance alone is what separates the best traders from the worst, our own experience in the markets leads us to believe that financial trading is indeed a skill, although it might be difficult at times to draw the line between talent and luck.

If successful trading is a skill, then it should be possible to automate it. Much of what traders do is basically analyse financial data, and try to find any information that will give them an edge over their competitors. Artificial intelligence could be very useful for that purpose; for instance, we could use data mining models to find the most intricate patterns in the data, and then utilize those patterns to make price predictions. This usage for data mining is not a novel idea; there are hundreds of studies on the subject, many of which were discussed in this dissertation. For the most part, these studies present empirical evidence indicating that data mining algorithms may be able to discover profitable patterns in financial data. It is possible that some of these results could be biased due to improper data mining technique, such as overfitting the test data, or not using enough instances to test the models. It is also possible that the existence of reviewer bias towards positive results (meaning articles that support the usefulness of data mining are more likely to get published than articles with negative results) could be distorting our view of the true potential of financial data mining. Regardless, the current consensus in literature is that these AI tools are well suited for the development of trading strategies, although it is unclear how well these strategies will work in the long run, or how profitable they really are when practical issues are taken into account.

Since we wanted to put computers in charge of financial trading, it became obvious from the beginning that we had to use data mining models to interpret the financial data. Our objective was not to turn them into supporting tools for human traders; rather, we wanted to completely replace these traders with their software equivalent. More specifically, we wanted to devise a way to create intelligent agents that could trade successfully. Financial markets are the perfect setting for deploying agent technology: it is easy to envision an intelligent agent replacing a human trader, given that it can parse data much faster, and should also be more reliable, due to being emotionless (i.e., not affected by fear or greed); besides, a software agent is cheaper, never gets sick or complains, and never deliberately engages in fraudulent activities that might harm its employer. Hence, on paper, the creation of trading agents seemed like a good idea. There are countless strategies one could use to implement this type of agent; after looking at the current state of the art, we decided it would be best to design a new agent architecture from scratch, specifically for the purpose of trading financial instruments. Initially, we came up with a very simple architecture (Barbosa & Belo, 2009a), which employed just one data mining model to make price direction forecasts for the USD/JPY currency pair. Then, we implemented an agent in accordance with that architecture; it predicted the direction of the price with the model, and then used the predictions to open trades in the Forex market. We quickly realized that this agent would not be resilient enough to trade successfully over a long period of time, because it could not adapt to new market conditions, and was not able to learn from its errors; also, because it relied on just one data mining model that was trained with a fixed set of data, it was clear that it would probably fail if the new financial data was very different from the training set. In order to improve the architecture, we designed a prediction mechanism that used an ensemble of data mining models to derive the price direction forecasts (Barbosa & Belo, 2008b). We tested several models, among which the similarity and the distance to average classifiers, which resulted from our research (Barbosa & Belo, 2009b). Upon noticing that some models were better at predicting price increases, while others were better at predicting price decreases, we decided to make the prediction mechanism aggregate the models' forecasts by attributing a different weight to each of them, according to their profitability in the recent past. By doing so, the agent should be capable of adapting to different market dynamics: as time goes by, the predictions of the worst models become less relevant, while the most profitable models become more important. Next, since we wanted the architecture to give rise to agents that could learn new patterns over time, we modified it so that the models in the ensemble were periodically retrained with new data. Ergo, the prediction mechanism was no longer just a simple set of immutable models; by automatically updating the models, the agent was able to learn new patterns from the data, even as it traded.

With the completion of the prediction module, our architecture had taken care of the problem of deciding when to buy or short sell a financial instrument. Nevertheless, to create an intelligent agent that could completely emulate the activity of a human trader, we still had to address other issues. For example, the agent needed to decide how much to invest in each trade – concretely, it should increase or decrease the investment amount, based on how safe it perceived each trade would be. A new module was inserted in the architecture to take care of this requirement. We got the idea for it when we analysed the results of the prediction module. After inspecting its forecasts for the out-of-sample data, we could see that the trades were more likely to be profitable when the predictions of the models in the ensemble were more consensual. More importantly, the amount of profit was also bigger when there was greater agreement in the models' predictions. To capitalize on these discrepancies, we came up with the empirical knowledge module, which is basically just a wrapper around a case-based reasoning system. Its function was to suggest the size of future trades, based on the returns of similar trades in the past (i.e., trades for which the data mining models made the exact same forecasts); if the similar trades were mostly profitable, the trade size was increased, otherwise it was decreased. We had satisfying results with the new agent; yet, it still required one last improvement. Just like human traders get taught some basic rules by senior traders, we needed a way to teach the intelligent agents directly. The obvious choice here was to integrate a rule-based expert system into the architecture; the module encapsulating this system was named domain knowledge module. With this last piece, our agent architecture had taken its final shape. Theoretically, it should allow the intelligent trading agents to achieve all the objectives that were set for them, namely the ability to adjust to changes in the environment, to keep learning as time goes by, and to stop trading when in adverse conditions.

To put this architecture to the test, we utilized it to create various trading agents. To facilitate the process, we started by implementing a software shell – the iQuant software – that embodied the inner workings of the architecture, and made it easier to develop new agents (Barbosa & Belo, 2008c). We used this software to train 10 agents meant for trading currency pairs in the Forex market with a 6-hour time frame. Their settings were chosen randomly by an automatic procedure; this is not the most appropriate way to do the configuration, but we felt it was the best option we had, because it enabled us to create agents much faster (compared to manually tweaking them), and we did not need to worry about inadvertently "over-optimizing" them, and biasing the results. These 10 agents were tested with a couple of years' worth of out-of-sample data, and their overall performance – measured in terms of return and maximum drawdown – was found to be acceptable. However, their accuracy was, in general, very disappointing. This was not surprising: even if the markets are not completely efficient, it will still be extremely difficult to predict the movements in financial instruments' prices, because there are too many uncontrollable variables affecting them (especially at shorter time frames, like the one the Forex agents were using). Fortunately, the low accuracy turned out not to be a very significant problem. The main objective of the proposed agent architecture is to optimize the profit, not the accuracy, because profit is the ultimate goal in financial trading. Seeing that all the Forex agents were profitable (before expenses) at the end of the simulation period, even without being very accurate, meant that they were capable of predicting the most important trades, i.e., those in which the price movements were bigger. Hence, overall, it may be concluded that these agents fulfilled their purpose. But that is not to say that they were flawless; on the contrary, their returns were too volatile, which indicates that there was a lot of risk associated with their individual strategies.

In order to decrease the risk, we implemented a diversified investment strategy using the 10 Forex agents (Barbosa & Belo, 2009c). It achieved a promising gross profit and maximum drawdown under simulation. However, we soon realized that, in real life, the trading expenses would severely affect the profitability of this strategy. These expenses are bound to impact the performance of any trader, human or computer, because they lower the probability that the trades will be profitable; that is the reason why "bad" strategies (i.e., those that are no better than random trading) have a negative expected return. When we looked at the trading statistics of the Forex agents, it became clear that the adverse impact of the trading costs was exacerbated by their short investment time frame – since they did not allow enough time for the instruments' prices to vary significantly, their potential profit per trade was too low compared to the commissions they had to pay, and so the risk/reward ratio of their trades was too high. Clearly, the best solution to this problem would be to extend the trading time frame – this would increase the range of the prices, thus leading to bigger potential profits for the same trading fees. Longer time frames might also improve the accuracy of the agents, because there will be less noise in the price data. Nonetheless, we decided to keep the shorter time frame, and set out to reduce the agents' expenses by cutting the number of trades they were making. Given certain specificities of the Forex market, we knew the agents would be able to eliminate many redundant trades by simply being aware of each other's decisions. This was the perfect setting for implementing a multi-agent system (Barbosa & Belo, 2010a). In order to do that, we came up with an agent communication language and a negotiation protocol which allowed them to interact; the resulting multi-agent system performed much better in testing than the diversified investment strategy that had no inter-agent communication, and in doing so proved empirically that the Forex agents were indeed talented, and may actually be useful in practice.

To study the suitability of the proposed architecture for participating in other markets, we utilized it to develop stock trading agents. We started with the implementation of 25 agents, which were also integrated in a multi-agent system (Barbosa & Belo, 2010b). The main reason for grouping the agents in the system was to lower the trading risk, through investment diversification; this was an absolute necessity, because the big variance in their individual returns, as well as the well-known limitations of backtesting results compared to real life trading, meant that it would be far too risky to let any of them trade on their own. As expected, the simulation results of the multi-agent system showed a considerable improvement over the individual performances of the 25 trading agents, both risk-wise and profit-wise. This confirmed that proper investment diversification was an essential requirement, if we were to develop an investment system with the potential to be consistently profitable in the long run.

Despite the encouraging results that our stock trading system got in the simulation period, the fickleness that characterises stock markets (or any other financial market, for that matter) makes us wary of drawing any decisive conclusions regarding the practical usefulness of the trading agents. That is the reason why we created a public website, where the predictions and the trading statistics of the 25 agents can be followed in real-time. This website has been available since 2009; so far, the performances of the agents have been consistent with their results in the simulation period. As a follow-up to this work, we created an even bigger multi-agent investment strategy, consisting of 60 stock trading agents (Barbosa & Belo, 2011); their forecasts and trading decisions are also available online[14].

[14] The 60 stock trading agents may be followed at http://ruibarbosa.eu/iquant/iquant_all.html.

Besides testing the architecture with currency and stock prices, we also tested it with index data. We trained several agents with this data, and had them trade over long periods of time, using different time frames. The results of these investment simulations were mixed: some agents traded profitably for the duration of the simulation, while others did not do very well. This experiment revealed a few shortcoming in our method for creating the agents: it showed that the proposed architecture was not well suited for certain instruments and time frames, and that the strategy we employed to do their configuration was not very effective – clearly, doing it randomly with an automatic mechanism was not the optimal way to achieve the best performances, because the resulting agents were not very optimized.

In the final part of our interdisciplinary study, we introduced a solution that brought the fields of finance and agency closer together. Building on top of our previous work, we developed a multi-agent system that could act as a real life "intelligent" hedge fund, requiring (almost) no human intervention. This system consists of 35 agents, trading in two different markets. It performed quite well in our investment simulation, but the only way to prove its usefulness would be to test it in the real markets for an extended period of time. The development of this multi-agent system was the next logical step, following our initial objective to devise a method of implementing intelligent agents able to act as financial traders. In regard to this objective, we believe we achieved it in a satisfying manner: the agent architecture we proposed did indeed allow for the creation of trading agents whose actions resemble those of successful human traders; in our experiments, many of these agents seemed capable of adapting to sudden changes in market conditions, which is the single most important skill when working in such uncertain environments. Furthermore, by grouping the agents together, we demonstrated that they can be utilized as the basis for the development of autonomous investment funds that have the potential to yield a reasonable return, with relatively low risk. Keeping in mind the dangers of extrapolating from past performance to future returns, and also the pitfalls in evaluating trading systems through backtesting, we believe the results of our experiments were good enough to warrant given our systems the chance to trade live with real funds. Figure 102 shows an updated screenshot of the iQuant website as of February of 2011, that displays the latest performance of the stock trading system (with 25 agents); notice that a big part of these results were obtained with an automatic forward-testing mechanism, which anyone can follow live. The chart contains three time series: the gross and net cumulative returns of the system (using 4:1 leverage), and the return that would be obtained by buying and holding the 25 stocks. It is clear that the system's performance is much better than that of the buy-and-hold strategy. Moreover, the series with the system's gross cumulative return is everything one could hope for: it has a pronounced upward slope, with very low volatility. This curve measures the "talent" of the trading agents; for more than 5 years, they have been able to extract profit from the stock market very consistently. The net return curve, on the other hand, measures how well that talent might translate into real profits when trading live; we can verify that, even though the trading costs substantially decreased the overall return, the system's performance is still very interesting. We should mention that, in practice, it may be possible to approximate the net return curve to the gross

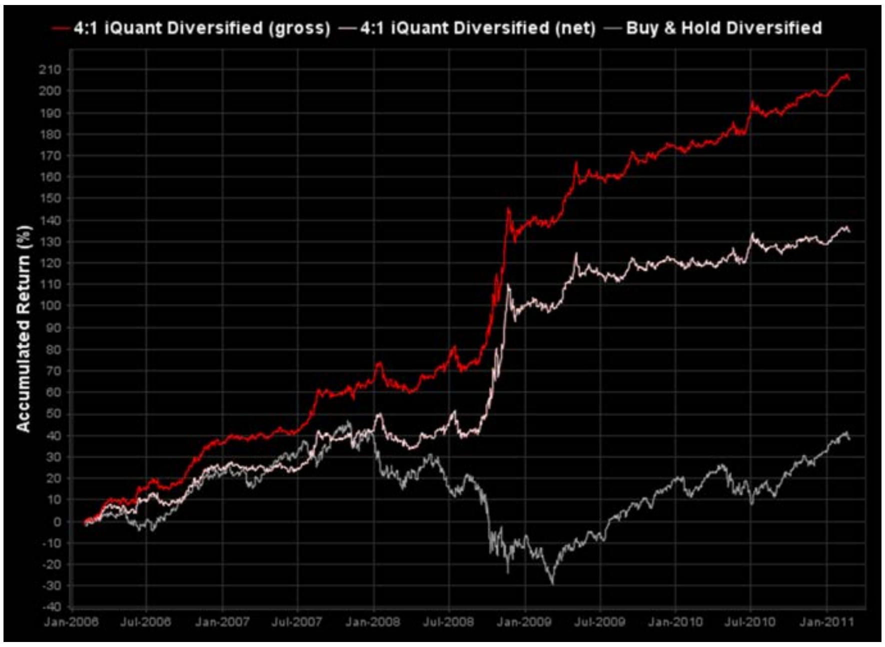


Figure 102. Updated gross and net cumulative returns of the stock trading system with 25 agents using 4:1 leverage.

return curve by optimizing the way the agents send their orders to the market, and ensuring that those orders are routed through the broker with the smallest fees. In our simulation environment, we are assuming commission costs that are 30% higher than those currently being charged at the cheapest online broker; this difference alone is a big drag on the system's net return. In any case, the forward-testing results indicate that the stock trading agents have been very competent thus far, and that their talent might yield real profits if the system is allowed to trade live.

Taking into account the results of all the experiments that were reported in this dissertation, we believe we have shown, beyond any doubt, that intelligent agents have a place in financial markets. In the process of doing so, we made several important contributions to different fields. Specifically, we submit that our research contributed to the advancement of the field of artificial intelligence in the following ways:

- we described a new model, the similarity classifier, that can be applied to any sort of data mining project;
- we devised a modified method of handling concept drift and ensemble online learning that is specifically customized for financial prediction (because the models' replacement and vote weights are based on their profit in the recent past);
- the prediction accuracy of the many agents that were tested may now be utilized as a benchmark for future studies on financial data mining;
- we presented a new way of integrating different AI solutions (data mining ensembles, case-based reasoning systems and expert systems) to create better systems that can exhibit more "intelligent" behaviour.

This last point may also be considered a contribution to the field of agency, because the systems we came up with are completely autonomous. Researchers in this field might also appreciate our work due to the following reasons:

- we proposed several new architectures for implementing intelligent trading agents, the last of which resulted in autonomous agents that were reasonably competent under simulation;
- we described a couple of negotiation protocols for the communication between the agents, and proved their importance by creating multi-agent systems that could perform much better than equivalent systems without inter-agent communication;
- future studies on automated investment systems may now use our agents' results as a basis for comparison; hence, researchers that develop new trading agent architectures will find it easier to evaluate the potential of their work;
- although we devised our agent architecture specifically for the task of trading financial instruments, it is possible that it could be adapted to other practical uses also requiring a mix of data mining and expert knowledge;
- we showed a novel practical application for agent technology; this is an important contribution, because this field has previously been criticized for the lack of deployed applications.

This last item pertains to the usefulness of our work in practice, which is what ultimately validated our research. Since we targeted a concrete problem in the field of finance, our experiments should also be relevant to researchers in that field, because:

- we demonstrated that intelligent agents have the potential to replace human traders;
- our initial plan to develop autonomous trading agents evolved to the point where we can envision full-size investment funds based solely on these agents; in this regard, we proposed the fusion of different multi-agent systems (acting in different markets) to create autonomous hedge funds, a concept that should interest entrepreneurs in the financial services industry;

- the trading statistics of our multi-agent systems clarified what may reasonably be expected from autonomous investment systems, which is something that was not easy to gauge when we started our work; these results can now be used as a benchmark for traditional investment companies, or for other "intelligent" funds;
- this dissertation lists all the requirements that must be fulfilled, in order to create a study on automated trading with real practical value; these include avoiding all sorts of biases, accounting for sensible trading expenses, using test data that includes periods of very low and very high volatility, etc.; also, we put forward that studies on this subject should always be accompanied with a mechanism to "forward-test" the systems over an extended period of time, as this is the only way to verify that the backtesting results are not biased.

The most significant conclusion that can be drawn from our research is that the application of agent technology in financial markets holds enormous potential. It is important to remember that the performance of our agents was limited by several self-imposed restrictions. For example, our one-size-fits-all method for picking their settings (which we adopted to avoid pitfalls associated with excessive tweaking) is clearly not the best way to do the configuration; we believe that much better returns could be obtained if each agent was individually fine-tuned, according to the financial instrument it was meant to trade. This would imply manually selecting the data mining models in its prediction module, and picking the most appropriate investment time frame, based on the price volatility of the financial instrument. Also, we could try to improve the agents' accuracy by using better attributes to train them. In our experiments, we only used attributes that could be applied to all types of financial instruments; it is likely that utilizing features that better characterise the investment vehicles – like interest rate differentials for currency pairs, or profit-to-earnings ratios for stocks – would improve the precision (although high accuracy is not a requirement for successful trading, it certainly would not hurt). Defining different expert rules in each agent's domain knowledge module could also prove beneficial; for instance, stock trading agents could be prevented from trading on ex-dividend dates, or when the corresponding companies were about to release their earnings; a better take-profit rule (and possibly a stop-loss rule) could also be specified for each agent, instead of using the same equation to calculate their profit targets. Finally, the tweaking of their empirical knowledge modules might also improve the overall return; we could allow the best performing agents to invest the maximum amount more often, by lowering the thresholds, and rather than having them all use the same standard trade size, we could assign bigger amounts to the best agents, and let them compound the gains and trade with borrowed funds. To improve the investment diversification, it would be important to experiment with different financial instruments and class attributes; for example, instead of training all the agents to predict the direction of the price, we could make some of them forecast the expected variance – this would allow us to implement option trading agents, which would trade volatility, rather than the price. One last thing to consider is that, just like in real life, some trading agents may eventually need to be "fired", and replaced with better agents. While we could not do this in our experiments, to avoid survivorship bias, it is definitely something that should be considered in live trading systems. This all goes to show that there is still a lot of room for improvement in the agents and the multi-agent systems that were presented, and possibly in the agent architecture that we proposed. And that is, we believe, the highlight of our work: we have undoubtedly demonstrated that the usage of intelligent agents for the purpose of trading financial instruments holds a lot of promise. Specifically, our research could open the door to the development of agent-based autonomous hedge funds consisting of hundreds of individually fine-tuned trading agents, each with a specific time frame and financial instrument combination, so as to maximize the investment diversification.

All things considered, we think the goal that was set for our research has been fulfilled. We clearly showed that intelligent agents belong in financial markets, and meticulously described one possible method for implementing them. This conclusion is likely to be of interest to the investment community at large.

References

Abraham, A. (2005). Hybrid Soft and Hard Computing Based Forex Monitoring Systems. Studies in Fuzziness and Soft Computing, Vol. 181, pp. 113-129.

Abraham, A., Philip, N., & Saratchandran, P. (2003). Modeling Chaotic Behavior of Stock Indices Using Intelligent Paradigms. International Journal of Neural, Parallel & Scientific Computations, Vol. 11, pp. 143-160.

Aha, D. (1992). Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. International Journal of Man-Machine Studies, Vol. 36, No. 2, pp. 267-287.

Aha, D., Kibler, D., & Albert, M. (1991). Instance-Based Learning Algorithms. Machine Learning, Vol. 6, pp. 37-66.

Aite Group (2008). The World According to Quants: From Alpha Discovery to Execution. Retrieved from http://www.aitegroup.com/Reports/ReportDetail.aspx?recordItemID=411.

Aizerman, M., Braverman, E., & Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control, Vol. 25, pp. 821-837.

Anderson, J. (1996). ACT: A Simple Theory of Complex Cognition. American Psychologist, Vol. 51, No. 4, pp. 355-365.

Andriyashin, A., Härdle, W., & Timofeev, R. (2008). Recursive Portfolio Selection with Decision Trees. Collaborative Research Center 649, Discussion Paper 2008-009. Retrieved from http://sfb649.wiwi.hu-berlin.de/papers/pdf/SFB649DP2008-009.pdf.

Angel, J., & McCabe, D. (2009). The Ethics of Speculation. Journal of Business Ethics, Vol. 90, No. 3, pp. 277-286.

Arnuk, S., & Saluzzi, J. (2009). What Ails Us About High Frequency Trading?. Retrieved from http://www.themistrading.com/article_files/0000/0508/What_Ails_Us_About_High_Frequency_Trading_--_Final__2__10-5-09.pdf.

Balvers, R., Wu, Y., & Gilliland, E. (2000). Mean Reversion across National Stock Markets and Parametric Contrarian Investment Strategies. The Journal of Finance, Vol. 55, No. 2, pp. 745-772.

Baker, S. (2011). Final Jeopardy: Man vs. Machine and the Quest to Know Everything. Houghton Mifflin Harcourt, Boston.

Barbosa, J., & Torgo, L. (2006). Online ensembles for financial trading. Proceedings of the Workshop on Practical Data Mining: Applications, Experiences and Challenges.

Barbosa, R., & Belo, O. (2008a). An Intelligent USD/JPY Trading Agent. Proceedings of the Adaptive and Learning Agents and Multi-Agent Systems Workshop at AAMAS'08, pp. 1-7. (ALAMAS+ALAg Workshop @ Estoril 2008)

Barbosa, R., & Belo, O. (2008b). Autonomous Forex Trading Agents. Advances in Data Mining: Medical Applications, E-Commerce, Marketing, and Theoretical Aspects, Springer Berlin / Heidelberg, pp. 389-403. (8th Industrial Conference on Data Mining – ICDM @ Leipzig 2008)

Barbosa, R., & Belo, O. (2008c). Algorithmic Trading Using Intelligent Agents. Proceedings of the 2008 International Conference on Artificial Intelligence, CSREA Press, pp. 136-142. (ICAI @ Las Vegas 2008)

Barbosa, R., & Belo, O. (2009a). A Step-By-Step Implementation of a Hybrid USD/JPY Trading Agent. International Journal of Agent Technologies and Systems, IGI Publishing, Vol. 1, No. 2, pp. 19-35.

Barbosa, R., & Belo, O. (2009b). Lazy Classification Using an Optimized Instance-Based Learner. Intelligent Data Engineering and Automated Learning, Springer Berlin / Heidelberg, pp. 66-73. (IDEAL @ Burgos 2009)

Barbosa, R., & Belo, O. (2009c). A Diversified Investment Strategy Using Autonomous Agents. Advances in Data Analysis, Data Handling and Business Intelligence, Springer Berlin / Heidelberg, pp. 339-350. (32nd Annual Conference of the German Classification Society – GfKl @ Hamburg 2008)

Barbosa, R., & Belo, O. (2010a). Multi-Agent Forex Trading System. Agent and Multi-Agent Technology for Internet and Enterprise Systems, Springer Berlin / Heidelberg, pp. 91-118.

Barbosa, R., & Belo, O. (2010b). The Agent-Based Hedge Fund. Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, IEEE Computer Society Press, Vol. 2, pp. 449-452. (WI-IAT @ Toronto 2010)

Barbosa, R., & Belo, O. (2010c). A Step-By-Step Implementation of a Multi-Agent Currency Trading System. Developments in Intelligent Agent Technologies and Multi-Agent Systems: Concepts and Applications, IGI Publishing, pp. 213-253.

Barbosa, R., & Belo, O. (2011). An Agent Task Force for Stock Trading. Forthcoming. (9th International Conference on Practical Applications of Agents and Multi-Agent Systems – PAAMS @ Salamanca 2011)

Baum, L., & Petrie, T. (1966). Statistical Inference for Probabilistic Functions of Finite State Markov Chains. The Annals of Mathematical Statistics, Vol. 37, No. 6, pp. 1554-1563.

Bellifemine, F., Poggi, A., & Rimassa, G. (1999). JADE – A FIPA-compliant agent framework. Proceedings of the 4th International Conference on the Practical Applications of Agents and MultiAgent Systems, pp. 97-108.

Bentley, J. (1980). Multidimensional Divide and Conquer. Communications of the ACM, Vol. 23, No. 4, pp. 214-229.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. Scientific American, May 2001 Issue, pp. 1-4.

Bigus, J., Schlosnagle, D., Pilgrim, J., Mills, W., & Diao, Y. (2002). ABLE: A toolkit for building multiagent autonomic systems. IBM Systems Journal, Vol. 41, No. 3, pp. 350-371.

Blum, A., & Furst, M. (1997). Fast planning through planning graph analysis. Artificial intelligence. Vol. 90, pp. 281-300.

Bollen, J., Mao, H., & Zeng, X. (2010). Twitter mood predicts the stock market. Retrieved from http://arxiv.org/abs/1010.3003.

Bollen, N., & Pool, V. (2009). Do Hedge Fund Managers Misreport Returns? Evidence from the Pooled Distribution. The Journal of Finance, Vol. 64, No. 5, pp. 2257-2288.

Booker, L., Goldberg, D., & Holland, J. (1989). Classifier systems and genetic algorithms. Artificial Intelligence, Vol. 40, pp. 235-282.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. Proceedings of the 5th Annual Workshop on Computational Leaning Theory, pp. 144-152.

Box, G., & Jenkins, G. (1976). Time Series Analysis: Forecasting and Control. Holden-Day, San Francisco.

Bratman, M., Israel, D., & Pollack, M. (1988). Plans and Resource-Bounded Practical Reasoning. Computational Intelligence, Vol. 4, No. 4, pp. 349-355.

Breiman, L. (1996). Bagging Predictors. Machine Learning, Vol. 24, No. 2, pp. 123-140.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and Regression Trees. Chapman & Hall, London.

Brooks, R. (1990). Elephants don't play chess. Robotics and Autonomous Systems, Vol. 6, pp. 3-15.

Brunnermeier, M. (2009). Deciphering the Liquidity and Credit Crunch 2007–2008. Journal of Economic Perspectives, Vol. 23, No. 1, pp. 77-100.

Buchanan, M. (2009). Meltdown modelling: Could agent-based computer models prevent another financial crisis?. Nature, Vol. 460, No. 6, pp. 680-682.

Buffett, W. (1984). The Superinvestors of Graham-and-Doddsville. Hermes, Fall 1984 Issue, pp. 4-15.

Castro, P., & Sichman, J. (2009). AgEx: A Financial Market Simulation Tool for Software Agents. Enterprise Information Systems, Vol. 24, pp. 704-715.

Cessie, S., & Houwelingen, J. (1992). Ridge Estimators in Logistic Regression. Applied Statistics. Vol. 41, No. 1, pp. 191-201.

Chang, F., Lin, C., & Lu, C. (2006). Adaptive Prototype Learning Algorithms: Theoretical and Experimental Studies. Journal of Machine Learning Research, Vol. 7, pp. 2125-2148.

Chen, S., & Yeh, C. (2001). Evolving traders and the business school with genetic programming: A new architecture of the agent-based artificial stock market. Journal of Economic Dynamics and Control, Vol. 25, pp. 363-393.

Chen, W., Shih, J., & Wu, S. (2006). Comparison of support-vector machines and back propagation neural networks in forecasting the six major Asian stock markets. International Journal of Electronic Finance, Vol. 1, No.1, pp. 49-67.

Chortareas, G., Kapetanios, G., & Shin, Y. (2002). Nonlinear Mean-Reversion in Real Exchange Rates. Economics Letters, Vol. 77, No. 3, pp. 411-417.

Chou, S., Yang, C., Chen, C., & Lai, F. (1996). A Rule-based Neural Stock Trading Decision Support System. Proceedings of the IEEE/IAFE 1996 Conference on Computational Intelligence for Financial Engineering, pp. 148-154.

Choudhry, R., & Garg, K. (2008). A Hybrid Machine Learning System for Stock Market Forecasting. Proceedings of World Academy of Science, Engineering and Technology, Vol. 29, pp. 315-318.

Cleary, J., & Trigg, L. (1995). K*: An Instance-based Learner Using an Entropic Distance Measure. Proceedings of the 12th International Conference on Machine Learning, pp. 108-114.

Cohen, W. (1995). Fast Effective Rule Induction. Proceedings of the 12th International Conference on Machine Learning, pp. 115-123.

Davidson, C. (1997). Trust me, I'm an expert. New Scientist, Issue 2111, pp. 26-31.

Demiröz, G., & Güvenir, H. (1997). Classification by Voting Feature Intervals. Proceedings of the 9th European Conference on Machine Learning, pp. 85-92.

Dichev, I., & Yu., G. (2010). Higher Risk, Lower Returns: What Hedge Fund Investors Really Earn. Journal of Financial Economics. Retrieved from http://ssrn.com/abstract=1354070.

Dickersin, K. (1990). The existence of publication bias and risk factors for its occurrence. The Journal of the American Medical Association, Vol. 263, No. 10, pp. 1385-1389.

Dreyfus, H. (1979). What Computers Can't Do. MIT Press, Cambridge.

Duch, W., Oentaryo, R., & Pasquier, M. (2008). Cognitive Architectures: Where do we go from here?. Frontiers in Artificial Intelligence and Applications, Vol. 171, pp. 122-136.

Duhigg, C. (2006, November 23). Artificial intelligence applied heavily to picking stocks. The New York Times. Retrieved from:
http://www.nytimes.com/2006/11/23/business/worldbusiness/23iht-trading.3647885.html.

Fama, E. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. The Journal of Finance, Vol. 25, No. 2, pp. 383-417.

Farmer, J., & Foley, D. (2009). The Economy Needs Agent-Based Modelling. Nature, Vol. 460, No. 6, pp. 685-686.

Fayyad, U., & Irani, K. (1993). Multi-interval discretization of continuous valued attributes for classification learning. Proceedings of the 13th International Joint Conference on Artificial Intelligence, pp. 1022-1027.

Fayyad, U., Shapiro, G., Smyth, P., & Uthurusamy, R. (1996). Advances in Knowledge Discovery and Data Mining. MIT Press, Cambridge.

Feigenbaum, E., Buchanan, B., & Lederberg, J. (1971). On Generality and Problem Solving: A Case Study Using the DENDRAL Program. Machine Intelligence, Vol. 6, pp. 165-190.

Ferguson, I. (1992). TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents. PhD thesis, University of Cambridge.

Finin, T., Fritzson, R., Mckay, D., & McEntire, R. (1994). KQML as an Agent Communication Language. Proceedings of the 3rd International Conference on Information and Knowledge Management, pp. 456-463.

FIPA (2002). FIPA ACL Message Structure Specification. FIPA TC Communication. Retrieved from http://www.fipa.org/specs/fipa00061/.

Fisher, M. (1994). A survey of Concurrent METATEM - the Language and its Applications. Proceedings of the 1st International Conference on Temporal Logic, pp. 480-505.

Fox, J. (2009). The Myth of the Rational Market: A History of Risk, Reward, and Delusion on Wall Street. HarperCollins Publishers, New York.

Frank, E., & Witten, I. (1998). Generating Accurate Rule Sets Without Global Optimization. Proceedings of the 15th International Conference on Machine Learning, pp. 144-151.

Franklin, S. & Graesser, A. (1996). Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. Proceedings of the Workshop on Intelligent Agents III: Agent Theories, Architectures, and Languages, pp. 21-35.

Franses, P. (1998). Time series models for business and economic forecasting. Cambridge University Press, Cambridge.

Fraser, A., & Burnell, D. (1970). Computer Models in Genetics. McGraw-Hill, New York.

Freund, Y., & Mason, L. (1999). The Alternating Decision Tree Learning Algorithm. Proceedings of the 16th International Conference on Machine Learning, pp. 124-133.

Freund, Y., & Schapire, R. (1999). Large margin classification using the perceptron algorithm. Machine Learning, Vol. 37, No. 3, pp. 277-296.

Friedlander, D., & Franklin, S. (2008). LIDA and a Theory of Mind. Frontiers in Artificial Intelligence and Applications, Vol. 171, pp. 137-148.

Friedman, J., Bentley, J., & Finkel, R. (1977). An Algorithm for Finding Best Matches in Logarithmic Expected Time. ACM Transactions on Mathematical Software, Vol. 3, No. 3, pp. 209-226.

Gaines, B., & Compton, P. (1995). Induction of Ripple-Down Rules Applied to Modeling Large Databases. Journal of Intelligent Information Systems, Vol. 5, No. 3, pp. 211-228.

Georgeff, M., & Lansky, A. (1987). Reactive Reasoning and Planning. Proceedings of the 6th National Conference on Artificial Intelligence, pp. 677-682.

Gobet, F., Lane, P., Croker, S., Cheng, P., Jones, G., Oliver, I., & Pine, J. (2001). Chunking mechanisms in human learning. TRENDS in Cognitive Sciences, Vol. 5, No. 6, pp. 236-243.

Grosan, C., & Abraham, A. (2006). Stock Market Modeling Using Genetic Programming Ensembles. Studies in Computational Intelligence, Vol. 13, pp. 131-146.

Hall, M. (1999). Feature selection for discrete and numeric class machine learning. Working paper, University of Waikato.

Hall, M. (2006). A Decision tree-based attribute weighting filter for naive Bayes. Working paper, University of Waikato.

Harries, M., & Horn, K. (1995). Detecting Concept Drift in Financial Time Series Prediction using Symbolic Machine Learning. Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence, pp. 91-98.

Harris, L. (1986). How to profit from intradaily stock returns. The Journal of Portfolio Management, Vol. 12, No. 2, pp. 61-64.

Hart, P., Nilsson, N., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, Vol. 4, No. 2, pp. 100-107.

Hasanhodzic, J., Lo, A., & Viola, E. (2009). A Computational View of Market Efficiency. Retrieved from http://arxiv.org/abs/0908.4580.

Hedge Fund Research (2009). Record Number of Hedge Funds Liquidate in 2008. Retrieved from http://www.hedgefundresearch.com/pdf/pr_20090318.pdf.

Helsinger, A., Thome, M., & Wright, T. (2004). Cougaar: a scalable, distributed multi-agent architecture. Proceedings of the 2004 IEEE International Conference on Systems, Man & Cybernetics, Vol. 2, pp. 1910-1917.

Hendler, J. (2007). Where Are All the Intelligent Agents?. IEEE Intelligent Systems, Vol. 22, No. 3, pp. 2-3.

Hindriks, K. (2001). Agent Programming Languages: Programming with Mental Models. PhD thesis, Utrecht University.

Hindriks, K., Boer, F., Hoek, W., & Meyer, J. (1999). Agent programming in 3APL. Autonomous Agents and Multi-Agent Systems, Vol. 2, No. 4, pp. 357-401.

Hirshleifer, D., & Shumway, T. (2003). Good Day Sunshine: Stock Returns and the Weather. The Journal of Finance, Vol. 58, No. 3, pp. 1009-1032.

Holmes, G., Hall, M., & Frank, E. (1999). Generating Rule Sets from Model Trees. Proceedings of the 12th Australian Joint Conference on Artificial Intelligence, pp. 1-12.

Hommes, C. (2006). Heterogeneous Agent Models in Economics and Finance. Handbook of Computational Economics, Vol. 2, pp. 1109-1186.

Horst, J., & Verbeek, M. (2007). Fund Liquidation, Self-selection, and Look-ahead Bias in the Hedge Fund Industry. Review of Finance, Vol. 11, pp. 605-632.

Horswill, I. (2000). Functional programming of behavior-based systems. Autonomous Robots, Vol. 9, pp. 83-93.

Jacobsen, B. & Visaltanachoti, N. (2009). The Halloween Effect in U.S. Sectors. Financial Review, Vol. 44, No. 3, pp. 437-459.

Jennings, N., & Wooldridge, M. (1998). Applications of Intelligent Agents. Agent Technology: Foundations, Applications, and Markets, pp. 3-28.

John, G., & Langley, P. (1995). Estimating Continuous Distributions in Bayesian Classifiers. Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence, pp. 338-345.

Jordan, J. (2010, December 22). Hedge Fund Will Track Twitter to Predict Stock Moves. Bloomberg. Retrieved from http://www.bloomberg.com/news/2010-12-22/hedge-fund-will-track-twitter-to-predict-stockmarket-movements.html.

Just, M., & Varma, S. (2007). The organization of thinking: What functional brain imaging reveals about the neuroarchitecture of complex cognition. Cognitive, Affective, & Behavioral Neuroscience, Vol. 7, No. 3, pp. 153-191.

Kaburlasos, V., Athanasiadis, I., & Mitkas, P. (2007). Fuzzy Lattice Reasoning (FLR) Classifier and its Application for Ambient Ozone Estimation. International Journal of Approximate Reasoning, Vol. 45, No. 1, pp. 152-188.

Kamruzzaman, J., & Sarker, R. (2003). Comparing ANN Based Models with ARIMA for Prediction of Forex Rates. ASOR Bulletin, Vol. 22, No. 22, pp. 2-11.

Kay, J. (2008, October 14). Banks got burned by their own 'innocent fraud'. Financial Times. Retrieved from http://www.ft.com/cms/s/0/12ade22e-99fc-11dd-960e-000077b07658.html.

Kearns, M., & Ortiz, L. (2003). The Penn-Lehman Automated Trading Project. IEEE Intelligent Systems, Vol. 18, No. 6, pp. 22-31.

Khandani, A., & Lo, A. (2007). What Happened To The Quants In August 2007?. MIT Laboratory for Financial Engineering. Retrieved from:
http://web.mit.edu/alo/www/Papers/august07.pdf.

Kim, K. (2003). Financial time series forecasting using support vector machines. Neurocomputing, Vol. 55, pp- 307-319.

Kim, M., Nelson, C., & Startz, R. (1991). Mean Reversion in Stock Prices? A Reappraisal of the Empirical Evidence. The Review of Economic Studies, Vol. 58, No. 3, pp. 515-528.

Kohavi, R. (1995a). A study of cross-validation and bootstrap for accuracy estimation and model selection. Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 1137–1143.

Kohavi, R. (1995b). The Power of Decision Tables. Proceedings of the 8th European Conference on Machine Learning, pp. 174-189.

Kohavi, R. (1996). Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid. Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, pp. 202-207.

Kohavi, R., Langley, P., & Yun, Y. (1997). The Utility of Feature Weighting in Nearest-Neighbor Algorithms. Proceedings of the Ninth European Conference on Machine Learning, pp. 85-92.

Kovalerchuk, B., & Vityaev, E. (2000). Data Mining in Finance: Advances in Relational and Hybrid Methods. Springer, New York.

Kurzweil, R. (2006). The Singularity is Near: When Humans Transcend Biology. Penguin Group, London.

Kwon, Y., & Moon, B. (2004). Evolutionary Ensemble for Stock Prediction. Genetic and Evolutionary Computation, pp. 1102-1113.

Laird, J., Newell, A., & Rosenbloom, P. (1987). Soar: An Architecture for General Intelligence. Artificial Intelligence, Vol. 33, No. 1, pp. 1-64.

Landwehr, N., Hall, M., & Frank, E. (2005). Logistic Model Trees. Machine Learning, Vol. 59, pp. 161-205.

LeBaron, B. (2006). Agent-Based Computational Finance. Handbook of Computational Economics, Vol. 2, pp. 1187-1233.

Lee, J., Park, J., O, J., Lee, J., & Hong, E. (2007). A Multiagent Approach to Q-Learning for Daily Stock Trading. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, Vol. 37, No. 6, pp. 864-877.

Lee, R. (2004). iJADE stock advisor: an intelligent agent based stock prediction system using hybrid RBF recurrent network. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, Vol. 34, No. 3, pp. 421-428.

Leeson, N., & Whitley, E. (1996). Rogue Trader: How I Brought Down Barings Bank and Shook the Financial World. Little, Brown and Company, New York.

Leinweber, D. (2007). Stupid Data Miner Tricks: Overfitting the S&P 500. The Journal of Investing, Vol. 16, No. 1, pp. 15-22.

Levesque, H., Reiter, R., Lespérance, Y., Lin, F., & Scherl, R. (1997). GOLOG: A Logic Programming Language for Dynamic Domains. Journal of Logic Programming, Vol. 31, pp. 59-83.

Li, J., & Tsang, E. (1999). Investment Decision Making Using FGP: A Case Study. Proceedings of the 1999 Congress on Evolutionary Computation, Vol. 2, pp. 1253-1259.

Luo, Y., Liu, K., & Davis, D. (2002). A multi-agent decision support system for stock trading. IEEE Network, Vol. 16, No. 1, pp. 20-27.

Mackworth, A. (1977). Consistency in networks of relations. Artificial Intelligence, Vol. 8, pp. 99-118.

Maes, P. (1991). The Agent Network Architecture (ANA). ACM SIGART Bulletin, Vol. 2, No. 4, pp. 115-120.

Maes, P. (1994). Agents that reduce work and information overload. Communications of the ACM, Vol. 37, No. 7, pp. 30-40.

Mahfound, S., & Mani, G. (1996). Financial forecasting using genetic algorithms. Journal of Applied Artificial Intelligence, Vol. 10, No. 6, pp. 543-565.

Malkiel, B. (1985). A Random Walk Down Wall Street. W. W. Norton & Company, New York.

Markram, H. (2006). The Blue Brain Project. Nature Reviews – Neuroscience, Vol. 7, pp. 153-160.

Martin, B. (1995). Instance-Based Learning: Nearest Neighbour with Generalisation. Master's thesis, University of Waikato.

Mauldin, M. (1994). ChatterBots, TinyMuds, and the Turing Test: Entering the Loebner Prize Competition. Proceedings of the 12th National Conference on Artificial Intelligence, Vol. 1, pp. 16-21.

McCarthy, J., Minsky, M., Rochester, N., & Shannon, C. (1955). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence. Retrieved from:
http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html.

McCool, G. (2010, January 27). Pressure rises on Galleon insider trade defendants. Reuters. Retrieved from:

http://www.reuters.com/article/2010/01/27/us-galleon-pleas-analysis-idUSTRE60Q6Q120100127.

Miller, M., Muthuswamy, J., & Whaley, R. (1994). Mean Reversion of Standard & Poor's 500 Index Basis Changes: Arbitrage-Induced or Statistical Illusion?. The Journal of Finance, Vol. 49, No. 2, pp. 479-513.

Minsky, M. (1967). Computation: Finite and Infinite Machines. Prentice Hall, New Jersey.

Mittermayer, M. (2004). Forecasting Intraday Stock Price Trends with Text Mining Techniques. Proceedings of the 37th Annual Hawaii International Conference on System Sciences.

Montgomery, D., & Peck, E. (1982). Introduction to linear regression analysis. Wiley, New Jersey.

Moody, J., & Darken, C. (1989). Fast Learning in Networks of Locally-Tuned Processing Units. Neural Computation, Vol. 1, No. 2, pp. 281-294.

Nash, J. (1950). Equilibrium points in n-person games. Proceedings of the National Academy of Sciences of the United States of America, Vol. 36, No. 1, pp. 48-49.

Nestor, A., & Kokinov, B. (2004). Towards Active Vision in the DUAL Cognitive Architecture. International Journal on Information Theories and Applications, Vol. 11, pp. 9-15.

Nwana, H., & Ndumu, D. (1999). A Perspective on Software Agents Research. The Knowledge Engineering Review, Vol. 14, No. 2, pp. 125-142.

Officer, A. (2009, January 5). The Ponzi Scheme in Every Hedge Fund. Time. Retrieved from http://www.time.com/time/business/article/0,8599,1869196,00.html.

Opitz, D., & Maclin, R. (1999). Popular Ensemble Methods: An Empirical Study. Journal of Artificial Intelligence Research, Vol. 11, pp. 169-198.

Palmer, R., Arthur, W., Holland, J., & LeBaron, B. (1999). An artificial stock market. Artificial Life and Robotics, Vol. 3, No. 1, pp. 27-31.

Panait, L., & Luke, S. (2005). Cooperative Multi-Agent Learning: The State of the Art. Autonomous Agents and Multi-Agent Systems, Vol. 11, No. 3, pp. 387-434.

Patterson, S. (2010, July 13). Letting the Machines Decide. The Wall Street Journal. Retrieved from http://online.wsj.com/article/SB10001424052748703834604575365310813948080.html.

Pettengill, G. (2003). A Survey of the Monday Effect Literature. Quarterly Journal of Business & Economics, Vol. 42, No. 3, pp. 3-28.

Poole, D., Mackworth, A., & Goebel, R. (1998). Computational Intelligence: A Logical Approach. Oxford University Press, Oxford.

Prokop, J. (2010). On the Persistence of a Calendar Anomaly: The Day-of-the-Week Effect in German and US Stock Market Returns. International Research Journal of Finance and Economics, No. 54, pp. 176-190.

Qi, M., & Zhang, G. (2008). Trend Time–Series Modeling and Forecasting With Neural Networks. IEEE Transactions on Neural Networks, Vol. 19, No. 5, pp. 808-816.

Quinlan, J. (1987). Simplifying decision trees. International Journal of Man-Machine Studies, Vol. 27, pp. 221-248.

Quinlan, J. (1992). Learning with Continuous Classes. Proceedings of the 5th Australian Joint Conference on Artificial Intelligence, pp. 343-348.

Quinlan, J. (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco.

Rijsbergen, C. (1979). Information Retrieval (2nd edition). Butterworths, London.

Robotti, C., & Krivelyova, A. (2003). Playing the Field: Geomagnetic Storms and the Stock Market. Federal Reserve Bank of Atlanta, Working Paper No. 2003-5b. Retrieved from http://www.frbatlanta.org/filelegacydocs/wp0305b.pdf.

Rodríguez, F., Rivero, S., & Félix, J. (1999). Exchange-rate forecasts with simultaneous nearest-neighbour methods: evidence from the EMS. International Journal of Forecasting, Vol. 15, No. 4, pp. 383-392.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review, Vol. 65, No. 6, pp. 386-408.

Rosenschein, J, & Zlotkin, G. (1994). Rules of Encounter: Designing Conventions for Automated Negotiation among Computers. MIT Press, Cambridge.

Rosenthal, R. (1979). The file drawer problem and tolerance for null results. Psychological Bulletin, Vol. 86, No. 3, pp. 638-641.

Rousseeuw, P., & Leroy, A. (1987). Robust regression and outlier detection. Wiley, New Jersey.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. Nature, Vol. 323, pp. 533-536.

Russel, S., & Norvig, P. (2002). Artificial Intelligence: A Modern Approach – 2nd Edition. Prentice Hall, New Jersey.

Saad, E., Prokhorov, D., & Wunsch, D. (1998). Comparative Study of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks. IEEE Transactions on Neural Networks, Vol. 9, No. 6, pp. 1456-1470.

Sadeh, N., Arunachalam, R., Eriksson, J., Finne, N., & Janson, S. (2003). TAC-03: A Supply-Chain Trading Competition. AI Magazine, Vol. 24, No. 1, pp. 92-94.

Samuelson, P. (1965). Proof That Properly Anticipated Prices Fluctuate Randomly. Industrial Management Review, Vol. 6, No. 2, pp. 41-49.

Schapire, R. (1990). The Strength of Weak Learnability. Machine Learning, Vol. 5, No. 2, pp. 197-227.

Schulenburg, S., & Ross, P. (2000). An Adaptive Agent Based Economic Model. Learning Classifier Systems: From Foundations to Applications, Vol. 1813, pp. 263-282.

Searle, J. (1980). Minds, Brains and Programs. Behavioral and Brain Sciences, Vol. 3, No. 3, pp. 417-457.

Searle, J. (2004). Mind: A Brief Introduction. Oxford University Press, Oxford.

Segal, R., & Kephart, J. (2000). Incremental Learning in SwiftFile. Proceedings of the 7th International Conference on Machine Learning, pp. 863-870.

Sehgal, V., & Song, C. (2007). SOPS: Stock Prediction using Web Sentiment. Proceedings of the 7th IEEE International Conference on Data Mining Workshops, pp. 21-26.

Sharpe, W. (1966). Mutual Fund Performance. Journal of Business, No. 39, pp. 119-138.

Shi, H. (2007). Best-first Decision Tree Learning. Master's thesis, University of Waikato.

Shiller, R. (1992). Market Volatility. MIT Press, Cambridge.

Simon, H. (1965). The Shape of Automation for Men and Management. Harper & Row, New York.

Sollich, P., & Krogh, A. (1996). Learning with Ensembles: How over-fitting can be useful. Advances in Neural Information Processing Systems, No. 8, pp. 190-196.

Sortino, F., & Price, L. (1994). Performance measurement in a downside risk framework. The Journal of Investing, Vol. 3, No. 3, pp. 59-64.

Sun, R., Merrill, E., & Peterson, T. (2001). From implicit skills to explicit knowledge: A bottom-up model of skill learning. Cognitive Science, Vol. 25, No. 2, pp. 203-244.

Swingler, K. (1996). Financial Prediction, Some Pointers, Pitfalls, and Common Errors. Neural Computing & Applications, Vol. 4, No. 4, pp. 192-197.

Tabb Group (2009). US Equity High Frequency Trading: Strategies, Sizing and Market Structure. Retrieved from http://www.tabbgroup.com/PublicationDetail.aspx?PublicationID=505.

Taleb, N. (2007). The Black Swan: The Impact of the Highly Improbable. Random House, New York.

Tay, F., & Cao, L. (2001). Application of support vector machines in financial time series forecasting. The International Journal of Management Science, Vol. 29, No. 4, pp. 309-317.

Tenti, P. (1996). Forecasting Foreign Exchange Rates Using Recurrent Neural Networks. Applied Artificial Intelligence, Vol. 10, No. 6, pp. 567-582.

TrimTabs (2009). How Hedge Fund Drop-Outs Affect Measurable Industry Returns. The Hedge Fund Flow Report, August Issue. Retrieved from http://www.barclayhedge.com/research/hedge-fund-flow-report/hedge-fund-flow-report.html.

Tsang, E., & Jaramillo, S. (2004). Computational Finance. IEEE Computational Intelligence Society Newsletter, August Issue, pp. 8-13.

Turing, A. (1950). Computing Machinery and Intelligence. Mind, Vol. 59, No. 236, pp. 433-460.

Volpe, R., & Dickson, L. (2004). A review of literature on how professional speculators view their role in financial markets and the capital formation process. Journal for Economic Educators, Vol. 4, No. 4, pp. 6-12.

Wang, Y., & Witten, I. (2002). Modeling for optimal probability prediction. Proceedings of the 19th International Conference in Machine Learning, pp. 650-657, 2002.

Weizenbaum, J. (1966). ELIZA - A Computer Program for the Study of Natural Language Communication Between Man And Machine. Communications of the ACM, Vol. 9, No. 1, pp. 36-45.

Wellman, M., Wurman, P., O'Malley, K., Bangera, R., Lin, S., Reeves, D., & Walsh, W. (2001). Designing the Market Game for a Trading Agent Competition. IEEE Internet computing, Vol. 5, No. 2, pp. 43-51.

Wilder, J. (1978). New Concepts in Technical Trading Systems. Trend Research, New York.

Williams, L. (1979). How I Made One Million Dollars Last Year Trading Commodities. Windsor Books, New York.

Wilson, D., & Martinez, T. (2000). Reduction Techniques for Instance-Based Learning Algorithms. Machine Learning, Vol. 38, No. 3, pp. 257-286.

Witten, I., & Frank, E. (2005). Data Mining: Practical machine learning tools and techniques - 2nd Edition. Morgan Kaufmann, San Francisco.

Wolf, M. (2008, March 18). Why Today's Hedge Fund Industry May Not Survive. Financial Times. Retrieved from:
http://www.ft.com/cms/s/0/c8941ad4-f503-11dc-a21b-000077b07658.html.

Wolpert, D. (1992). Stacked generalization. Neural Networks, Vol. 5, No. 2, pp. 241–259.

Wooldridge, M. (2002). An Introduction to MultiAgent Systems. Wiley, New Jersey.

Wooldridge, M., & Jennings, N. (1995). Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, Vol. 10, pp. 115-152.

Wu, S., & Lu, R. (1993). Combining artificial neural networks and statistics for stock-market forecasting. Proceedings of the 1993 ACM conference on Computer science, pp. 257-264.

Wu, X., Kumar, V., Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G., Ng, A., Liu, B., Yu, P., Zhou, Z., Steinbach, M., Hand, D., & Steinberg, D. (2007). Top 10 algorithms in data mining. Journal Knowledge and Information Systems, Vol. 14, No. 1, pp. 1-37.

Yamazaki, T., & Ozasa, S. (2011, April 21). Ex-Goldman Sachs Banker Starts Hedge Fund Analyzing Japanese Blog Traffic. Bloomberg. Retrieved from:
http://www.bloomberg.com/news/2011-04-21/ex-goldman-banker-starts-hedge-fund-analyzing-japanese-blogs.html.

Young, T. (1991). Calmar Ratio: A Smoother Tool. Futures, Vol. 20, p. 40.

Yu, L., Lai, K., & Wang, S. (2005a). Designing a Hybrid AI System as a Forex Trading Decision Support Tool. Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence, pp. 89-93.

Yu, L., Wang, S., & Lai, K. (2005b). Mining Stock Market Tendency Using GA-Based Support Vector Machines. Internet and Network Economics, pp. 336-345.

Yuan, K., Zheng, L., & Zhu, Q. (2006). Are investors moonstruck? Lunar phases and stock returns. Journal of Empirical Finance, Vol. 13, No. 1, pp. 1-23.

Zemke, S. (1998). Nonlinear Index Prediction. Physica A: Statistical Mechanics and its Applications, Vol. 269, pp. 177-183.

Zhang, Y., Akkaladevi, S., Vachtsevanos, G., & Lin, T. (2002). Granular neural web agents for stock prediction. Soft Computing - A Fusion of Foundations, Methodologies and Applications, Vol. 6, No. 5, pp. 406-413.

The URLs referenced in this section were available on May 1st, 2011.

Appendix

The table that follows contains the composition of the ensembles of all the agents mentioned in this dissertation. For each agent, the list of data mining models is provided. For each of these models (described in Section 4.1), we present the training parameters (as defined by the Weka API) and attributes (described in Section 4.2). This information should be enough to completely reproduce the experiments reported in the text.

Agent	Model	Parameters	Attributes	Prediction
CHFJPY	RBFNetwork	-B 2 -S 1 -R 1.3E-8 -M -1 -W 0.19	price direction, hour (nom), day of week (num), MA(32), LAG(6), WILRS(7), RSI(32), ROC(32)	Class
	SimpleCart	-S 3 -M 1 -N 5 -C 1.0	price direction, hour (num), MA(37), WILR(34), WILRS(13), WILRS(37), ROC(4), ROC(22)	Class
	NaiveBayes	-K	% price change, close price, LAG(5), LAG(8), WILRS(39), RSIS(16)	Class
	BFTree	-S 6 -M 7 -N 4 -G -C 1.0 -P POSTPRUNED	price direction, close price, hour (num), day of week (nom), MA(20), MA(34), WILRS(11), RSI(32)	Class
	ADTree	-B 11 -E -2 -D	% price change, day of week (nom), WILR(6), WILRS(13), RSI(7), ROC(3)	Class
	JRip	-F 3 -N 0.11 -O 4 -S 3	close price, hour (nom), day of week (num), LAG(8), WILR(4), WILRS(17), RSI(40)	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -Z	hour (num), day of week (nom), WILR(30)	% Change
EURCHF	JRip	-F 2 -N 3.35 -O 4 -S 2	% price change, hour (num), day of week (nom), WILR(24), RSI(27), RSI(39), RSIS(8)	Class
	Logistic	-R 0.05 -M -1	price direction, hour (num), day of week (nom), RSI(18), RSI(28)	Class
	SimpleCart	-S 5 -M 2 -N 4 -C 0.67	price direction, close price, hour (num), day of week (num), LAG(5), RSIS(7), ROC(20), ROC(21)	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -Z	% price change, RSI(9), RSI(25)	% Change
	KStar	-B 38 -M m	price direction, MA(11), LAG(1), WILR(8), ROC(1)	% Change
	RBFNetwork	-B 6 -S 5 -R 1.112E-8 -M -1 -W 0.12	price direction, hour (num), WILR(17), RSI(23), ROC(3)	% Change
	J48	-C 0.05 -B -M 42 -A	price direction, hour (num), day of week (nom), LAG(1), WILR(11), RSI(14)	Class
EURGBP	SimilarityClassifier	-S 0.28 -W correlation -C unsupervised	day of week (nom), RSIS(6), ROC(34)	Class
	LMT	-I 32 -M 34 -W 0.0 -A	hour (nom), MA(18), LAG(2)	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -Z -B	day of month, day of week (num), WILR(29), WILRS(28), RSI(18), RSIS(8), ROC(37)	% Change
	KStar	-B 47 -M a	hour (nom), day of week (num), MA(2), MA(10), LAG(4), ROC(12)	Class
	NaiveBayes		hour (nom), day of week (nom), price direction	Class

	LinearRegression	-S 2 -R 6.92E-8	hour (nom), day of month, month, RSIS(19), ROC(7), ROC(40)	% Change
	DecisionTable	-X 2 -S "weka.attributeSelection.BestFirst -D 1 -N 5"	hour (nom), day of week (nom), day of month, LAG(1), ROC(28)	Class
EURJPY	IB1		price direction, hour (nom), day of week (num), MA(8), MA(12), RSI(15)	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -Z	price direction, hour (num), day of week (nom), LAG(7), WILR(25), ROC(7)	% Change
	SimpleCart	-S 1 -M 3 -N 3 -U -H -C 0.98	hour (num), day of week (num), MA(8)	Class
	PART	-B -M 7 -C 0.19 -Q 8	price direction, hour (nom), day of week (num), RSI(21), ROC(7)	Class
	KStar	-B 17 -M m	price direction, hour (nom), day of week (num), WILR(11), ROC(7)	% Change
	KStar	-B 16 -M a	price direction, hour (nom), day of week (num), MA(8), RSI(12), RSI(20)	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -Z -B	hour (num), day of week (nom), MA(7)	% Change
EURUSD	NaiveBayes		hour (nom), day of week (nom), % price change	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -H -Z -B	hour (nom), day of week (num), MA(2), RSI(11), ROC(12)	% Change
	SimpleCart	-S 1 -M 2 -N 6 -H -C 0.99	hour (nom), day of week (nom), LAG(2), RSI(2), ROC(2), ROC(5)	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.0010 -P 0.1	hour (nom), day of week (nom), MA(6), MA(4), MA(3), % price change	% Change
	LeastMedSq	-S 2 -G 0	hour (nom), day of week (nom), LAG(5), LAG(4), LAG(3), LAG(2), LAG(1), % price change	% Change
	KStar	-B 10 -M a	hour (nom), day of week (nom), price direction	% Change
	RBFNetwork	-B 6 -S 6 -R 5.93E-8 -M -1 -W 0.358	% price change, hour (num), day of week (nom), MA(12), ROC(4)	Class
GBPCHF	IB1		% price change, hour (num), MA(7), LAG(2), RSI(29)	Class
	IB1		price direction, close price, hour (num), day of week (nom), LAG(8), RSI(6)	Class
	KStar	-B 39 -M n	price direction, hour (num), LAG(2), WILR(28), RSI(5), RSI(23)	Class
	SimpleCart	-S 5 -M 5 -N 5 -U -H -C 1.0	% price change, hour (num), MA(10), WILR(21)	Class
	SimpleCart	-S 1 -M 4 -N 6 -U -H -C 1.0	% price change, hour (num), day of week (nom), LAG(7), WILR(27), RSI(34), RSIS(4), RSIS(9)	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -H -B	% price change, hour (nom), day of week (num), LAG(1), WILRS(29), RSI(18)	% Change
	PART	-B -M 6 -C 0.33 -Q 3	price direction, close price, hour (nom), day of week (nom), WILR(21), ROC(19)	Class
GBPJPY	IB1		% price change, close price, day of week (num), MA(27), LAG(2), RSIS(15)	Class
	NaiveBayes	-K	% price change, LAG(5), LAG(6), ROC(8), ROC(10)	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -Z -B	hour (nom), day of month, day of week (num), WILRS(10), WILRS(16)	% Change
	IB1		price direction, hour (nom), LAG(1), ROC(7), ROC(10), ROC(11)	Class
	J48	-S -C 0.23 -M 11 -A	price direction, hour (nom), day of week (num), LAG(6), RSI(2), ROC(12)	Class
	PART	-B -M 6 -C 0.32 -Q 10	price direction, hour (num), day of week (num), MA(11), WILR(29), ROC(4)	Class

	SimpleCart	-S 8 -M 4 -N 3 -U -C 1.0	% price change, close price, LAG(6)	Class
GBPUSD	M5P	-M 7 -L	% price change, hour (nom), LAG(6), LAG(7)	% Change
	KStar	-B 27 -M d	% price change, hour (nom)	Class
	NaiveBayes	-K	hour (nom), day of week (nom), LAG(7)	Class
	IB1		day of week (num), MA(7), LAG(3), LAG(6), LAG(7), RSI(22), RSI(24), ROC(18)	Class
	LMT	-B -C -I 26 -M 20 -W 0.0 -A	% price change, hour (num), day of week (nom), LAG(6), RSIS(26)	Class
	NaiveBayes	-K	price direction, hour (nom), day of week (nom), day of month, LAG(7)	Class
	Logistic	-R 0.03 -M 8	hour (num), day of week (nom), month, WILRS(19), RSI(9), RSIS(6), RSIS(7), RSIS(17)	Class
USDCHF	RBFNetwork	-B 2 -S 10 -R 3.0933E-8 -M -1 -W 0.48	hour (num), LAG(6), WILRS(6), WILRS(34)	Class
	IBk	-K 39 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	hour (num), LAG(1), LAG(6), WILR(24)	% Change
	PaceRegression	-E pace6	hour (num), day of week (num), MA(4), LAG(4)	% Change
	LibSVM	-S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1	hour (nom), LAG(4), WILR(7), WILR(23)	Class
	IB1		close price, hour (nom), day of week (nom), LAG(6), WILR(31), RSIS(22), ROC(35)	Class
	SimpleCart	-S 4 -M 3 -N 3 -H -C 1.0	price direction, hour (num), WILR(14)	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -Z -B	hour (nom), day of week (num), MA(11), RSI(2), RSI(29)	% Change
USDJPY	KStar	-B 35 -M a	hour (nom), day of week (nom), MA(6), price direction	Class
	J48	-C 0.25 -M 2	hour (nom), day of week (nom), MA(6), price direction	Class
	JRip	-F 3 -N 2.0 -O 2 -S 1 -E	hour (nom), day of week (nom), price direction	Class
	NaiveBayes		hour (nom), day of week (nom), % price change	Class
	LMT	-I -1 -M 15 -W 0.0 -A	hour (nom), MA(6), price direction	Class
	KStar	-B 35 -M a	hour (nom), day of week (nom), MA(6), price direction	% Change
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.0010 -P 0.1	hour (num), day of week (num), MA(10), MA(2), % price change	% Change
AA	NaiveBayes		day of week (nom), MA(8)	Class
	LeastMedSq	-S 10 -G 0	% price change, close price, LAG(6), LAG(7), WILR(6), RSIS(8), RSIS(35), ROC(25)	% Change
	LinearRegression	-S 2 -R 7.71E-8	day of month, WILR(29)	% Change
	PaceRegression	-E subset	LAG(2), WILR(24), RSI(16)	% Change
	VotedPerceptron	-I 1 -E 4.211 -S 13 -M 18486	% price change, close price, month, MA(32)	Class
	IB1		price direction, month, day of week (num), WILR(22), WILRS(9)	Class
	IBk	-K 27 -W 13 -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	close price, day of month, month, day of week (num), WILR(25), WILRS(3)	Class
	ADTree	-B 11 -E 17 -D	day of month, month, day of week (num), MA(16), MA(20), MA(31)	Class
	REPTree	-M 13 -V 0.0012 -N 6 -S 7 -L -1	day of month, MA(11), LAG(3), WILR(20), RSI(35), RSIS(28), ROC(10), ROC(25)	Class

	SimpleCart	-S 3 -M 3 -N 6 -H -C 1.0	month, WILR(2), ROC(3)	Class
	Ridor	-F 4 -S 1 -N 7 -A	% price change, day of month, MA(27), WILR(24), WILR(39), WILRS(25), RSI(26), ROC(19)	Class
AAPL	NaiveBayes		close price, MA(30), RSI(2), RSI(5), ROC(33)	Class
	LinearRegression	-S 1 -R 1.27E-8	price direction, day of month, LAG(6), RSIS(21)	% Change
	Logistic	-R 0.03 -M -1	close price, day of week (nom), RSIS(22)	Class
	PaceRegression	-E nested	close price, month, MA(36)	% Change
	RBFNetwork	-B 4 -S 10 -R 8.9E-8 -M -1 -W 0.2	% price change, close price, LAG(4), RSIS(16), RSIS(32)	Class
	VotedPerceptron	-I 1 -E 1.21 -S 2 -M 5527	price direction, close price, day of week (nom), MA(4), MA(13), WILR(14), ROC(16), ROC(28)	Class
	IBk	-K 3 -W 0 -E -F -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	% price change, day of month, MA(3), MA(28), WILRS(23), RSIS(14)	% Change
	FLR	-R 0.58 -Y -B	% price change, close price, month, day of week (num), LAG(1), LAG(4), RSI(21), RSIS(14)	Class
	M5P	-M 15	% price change, month, day of week (num), WILR(8), RSI(7), RSI(23)	% Change
	SimpleCart	-S 4 -M 4 -N 3 -U -H -C 0.7	% price change, day of month, month, WILR(8), WILRS(28), RSI(29), ROC(28)	Class
	NNge	-G 8 -I 7	close price, day of month, MA(37), RSI(4), RSIS(18), ROC(19), ROC(22), ROC(27)	Class
ADBE	NaiveBayes	-K	close price, day of week (num), WILR(5), WILRS(12), RSI(2), RSI(8)	Class
	LeastMedSq	-S 8 -G 8	close price, day of month, day of week (num), ROC(7)	% Change
	PaceRegression	-E pace6	% price change, day of month, month, day of week (num), LAG(5), LAG(8), WILRS(9), RSIS(25)	% Change
	KStar	-B 7 -M m	price direction, day of week (num), MA(38), LAG(1), WILR(38), WILRS(10), ROC(6)	Class
	FLR	-R 0.589 -Y -B	close price, day of month, month, day of week (num), WILRS(14), ROC(34)	Class
	LMT	-I 34 -M 16 -W 0.394	day of week (nom), MA(8), RSIS(14), RSIS(22), RSIS(25)	Class
	M5P	-U -M 16 -L	day of week (num), RSIS(24), ROC(7)	% Change
	SimpleCart	-S 5 -M 2 -N 4 -U -H -C 1.0	% price change, day of month, month, day of week (num), MA(21), LAG(1)	Class
	ConjunctiveRule	-N 5 -M 7.46 -P 9 -S 1	day of month, LAG(4), WILRS(8), WILRS(21), WILRS(39), RSI(2), RSI(19), RSIS(11)	% Change
	NNge	-G 7 -I 4	price direction, day of week (nom), day of month, month, WILR(2)	Class
	Ridor	-F 3 -S 1 -N 7 -A	price direction, day of week (nom), day of month, WILR(4), RSIS(25)	Class
BAC	LeastMedSq	-S 2 -G 10	day of week (num), RSI(8), RSI(33), RSIS(31), RSIS(35), ROC(32)	% Change
	RBFNetwork	-B 6 -S 9 -R 7.29E-8 -M -1 -W 0.26	day of week (nom), day of month, month, WILR(16), RSIS(3), ROC(2)	% Change
	VotedPerceptron	-I 3 -E 1.92 -S 2 -M 13170	price direction, close price, day of week (num), LAG(3), WILRS(28), RSI(21)	Class
	IBk	-K 16 -W 13 -E -F -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	close price, month, day of week (num), MA(6), MA(31), LAG(3), WILR(23)	Class
	KStar	-B 7 -M d	price direction, close price, day of month, day of week (num), WILR(4), WILRS(39), RSI(20)	% Change

	FLR	-R 0.59 -B	day of week (num), WILRS(13), ROC(26)	Class
	VFI	-B 0.74	% price change, day of month, WILR(38), WILRS(33), RSI(2), RSI(23)	Class
	LMT	-B -P -I 3 -M 39 -W 0.0	% price change, close price, day of week (nom), day of month, month, MA(8), WILRS(13), WILRS(17)	Class
	M5P	-U -M 49	close price, day of week (num), RSIS(37), ROC(37)	% Change
	SimpleCart	-S 3 -M 1 -N 4 -U -H -C 1.0	price direction, close price, day of week (nom), WILR(23), RSIS(22)	Class
	ConjunctiveRule	-N 4 -M 7.53 -P -1 -S 1	day of week (num), MA(12), LAG(2), WILR(32), RSI(21), RSIS(2), RSIS(37), ROC(31)	Class
CAL	LeastMedSq	-S 5 -G 0	close price, month, MA(25), RSIS(11), ROC(25)	% Change
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -Z	month, WILR(21), RSIS(26)	% Change
	LinearRegression	-S 2 -C -R 4.44E-8	price direction, month, WILRS(25), RSI(37), RSIS(4), RSIS(11), RSIS(26), ROC(18)	% Change
	VotedPerceptron	-I 2 -E 3.24 -S 9 -M 10186	close price, day of month, month	Class
	KStar	-B 13 -M m	price direction, close price, day of month, month, LAG(3), WILR(2), ROC(26), ROC(27)	% Change
	VFI	-C -B 0.59	month, MA(14), MA(18), WILRS(27), RSI(23)	Class
	LMT	-B -P -I 33 -M 32 -W 0.0	price direction, day of week (nom), day of month, month, MA(3), WILRS(11), ROC(11)	Class
	M5P	-U -M 34	day of month, month, MA(15), WILR(38), RSI(18), RSI(28), ROC(30)	% Change
	REPTree	-M 2 -V 0.001 -N 4 -S 6 -L 29	% price change, day of month, MA(11), WILRS(11), WILRS(25)	Class
	JRip	-F 5 -N 2.26 -O 3 -S 1 -E	price direction, close price, day of month, month, WILR(11), ROC(10), ROC(13)	Class
	Ridor	-F 3 -S 1 -N 8 -A	% price change, day of month, month, day of week (num), LAG(8), WILR(15), WILRS(31), ROC(2)	Class
CSCO	LinearRegression	-S 1 -C -R 9.34E-8	% price change, day of week (nom), day of month, MA(9), RSIS(22), RSIS(33), ROC(4), ROC(22)	% Change
	IBk	-K 18 -W 19 -F -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	month, day of week (num), MA(34)	Class
	FLR	-R 0.54 -B	close price, month, WILR(18), WILR(27), WILRS(36), RSI(25), RSIS(31), RSIS(38)	Class
	VFI	-B 0.62	WILR(34), WILRS(9), RSIS(12)	Class
	ADTree	-B 13 -E -1	price direction, day of month, day of week (num)	Class
	BFTree	-S 8 -M 17 -N 6 -G -C 1.0 -P POSTPRUNED	MA(3), MA(6), LAG(3), WILR(19), WILR(23), WILR(33), RSI(8)	Class
	J48	-L -C 0.38 -M 48	price direction, day of week (nom), day of month, LAG(5), WILR(20), WILR(27)	Class
	REPTree	-M 15 -V 0.001 -N 6 -S 8 -L -1	month, WILRS(8), RSIS(7)	% Change
	SimpleCart	-S 1 -M 3 -N 4 -U -C 1.0	day of month, day of week (num), WILR(9), WILRS(4)	Class
	ConjunctiveRule	-N 5 -M 1.11 -P -1 -S 10	% price change, day of month, month, LAG(8), WILR(17), WILR(21), WILRS(9)	% Change
	PART	-B -M 10 -C 0.24 -Q 3	day of week (nom), day of month, LAG(5), WILR(24), RSI(40), RSIS(2)	Class
DELL	LeastMedSq	-S 7 -G 7	% price change, MA(39), WILR(3), WILR(11)	% Change

	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -H	price direction, day of week (nom), WILR(39)	% Change
	Logistic	-R 0.059 -M -1	% price change, close price, day of week (nom), MA(4), MA(27), WILR(26), RSI(27)	Class
	RBFNetwork	-B 4 -S 10 -R 7.1E-8 -M -1 -W 0.31	% price change, month, WILR(15), RSI(37), ROC(30)	% Change
	VotedPerceptron	-I 1 -E 2.8 -S 1 -M 19270	price direction, close price, day of month, WILR(22), WILRS(8), RSI(26)	Class
	KStar	-B 1 -M a	% price change, close price, day of month, WILRS(23), RSI(33)	Class
	BFTree	-S 5 -M 16 -N 8 -H -G -C 1.0 -P POSTPRUNED	price direction, day of week (nom), month, MA(2), WILRS(29)	Class
	LMT	-I 20 -M 23 -W 0.73	price direction, month, MA(15), WILR(21), RSI(30), RSI(34)	Class
	REPTree	-M 3 -V 0.001 -N 4 -S 5 -L -1	day of week (nom), day of month, MA(6), MA(30), RSIS(14)	% Change
	SimpleCart	-S 6 -M 1 -N 3 -U -C 0.81	close price, month, day of week (num), WILR(27)	Class
	JRip	-F 5 -N 5.75 -O 4 -S 5	price direction, close price, day of week (nom), day of month, ROC(13)	Class
DIS	NaiveBayes	-K	close price, day of month, day of week (num)	Class
	Logistic	-R 0.03 -M 5	day of week (nom), day of month, month, LAG(1), WILR(17), RSI(7), ROC(35)	Class
	VotedPerceptron	-I 2 -E 4.53 -S 4 -M 12972	day of week (nom), RSI(22), ROC(13)	Class
	IBk	-K 1 -W 4 -E -I -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	price direction, month, LAG(6), LAG(8), WILRS(19), WILRS(28), ROC(37)	Class
	KStar	-B 47 -E -M m	price direction, close price, MA(23), LAG(7), WILR(14), RSI(34)	Class
	BFTree	-S 7 -M 13 -N 4 -H -C 0.69 -P POSTPRUNED	% price change, close price, ROC(18), ROC(25), ROC(27)	Class
	J48	-L -R -N 3 -Q 1 -B -M 15 -A	% price change, day of week (nom), day of month, month, LAG(3), ROC(40)	Class
	NBTree		price direction, close price, day of month, day of week (num)	Class
	REPTree	-M 2 -V 6.21E-4 -N 2 -S 8 -L -1	close price, day of week (nom), day of month, LAG(7), ROC(17), ROC(22)	Class
	PART	-B -M 7 -C 0.25 -Q 7	day of week (num), RSI(7), RSI(35)	Class
	Ridor	-F 4 -S 1 -N 3 -A	% price change, day of week (num), MA(19), LAG(2), LAG(8), WILRS(31), RSI(30), RSI(33)	Class
GE	NaiveBayes	-K	close price, day of week (nom), day of month, MA(18), LAG(6), RSIS(5), ROC(20)	Class
	KStar	-B 36 -M n	price direction, day of month, day of week (num), MA(4), LAG(7), ROC(39)	% Change
	REPTree	-M 3 -V 0.002 -N 4 -S 4 -L -1	close price, MA(36), LAG(5), RSIS(40)	Class
	IBk	-K 13 -W 20 -F -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	price direction, day of month, day of week (num), WILR(29), RSI(39)	Class
	ADTree	-B 12 -E -1	close price, day of month, day of week (num), WILR(5), ROC(24)	Class
	Ridor	-F 5 -S 1 -N 8 -A	% price change, month, day of week (num), WILRS(36)	Class
	LMT	-B -C -I 6 -M 32 -W 0.73	WILRS(23), WILRS(25), RSIS(28)	Class
	M5P	-U -M 24 -L	close price, day of month, month, day of week (num), MA(7), LAG(5), WILR(6), RSI(27)	% Change

	PaceRegression	-E olsc -S 2.0	close price, LAG(1), LAG(3), RSI(16), RSI(19), RSIS(27)	% Change
	VFI	-C -B 0.732	day of month, day of week (num), LAG(1), WILR(6)	Class
	JRip	-F 2 -N 2.87 -O 3 -S 8 -E	close price, month, LAG(2), RSIS(7)	Class
GOOG	LeastMedSq	-S 8 -G 7	WILRS(14), RSI(18), RSI(35)	% Change
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -B	day of week (nom), month, WILR(19)	% Change
	LinearRegression	-S 2 -C -R 9.1E-8	MA(35), LAG(5), ROC(6)	% Change
	PaceRegression	-E pace6	% price change, month, MA(25), LAG(5), RSIS(33)	% Change
	KStar	-B 26 -E -M n	price direction, close price, month, day of week (num), MA(30), MA(34), WILR(27), RSIS(17)	Class
	BFTree	-S 4 -M 6 -N 5 -H -C 0.78 -P POSTPRUNED	price direction, RSIS(13), ROC(30)	Class
	LMT	-B -C -P -I 6 -M 20 -W 0.0	close price, WILRS(16), RSI(5), RSI(20), RSI(39)	Class
	M5P	-M 39 -L	close price, month, MA(9), LAG(5), RSI(30), ROC(23)	% Change
	ConjunctiveRule	-N 2 -M 6.75 -P -1 -S 3	price direction, close price, day of week (nom), day of month, WILR(3), WILRS(29), RSI(23), RSIS(17)	% Change
	JRip	-F 2 -N 7.365 -O 4 -S 9 -E -P	% price change, day of week (nom), day of month, LAG(2)	Class
	M5Rules	-M 7	LAG(5), WILRS(29), RSIS(29)	% Change
HD	NaiveBayes	-K	% price change, close price, day of month, month, LAG(6), RSI(36), RSIS(35), RSIS(39)	Class
	LinearRegression	-S 1 -R 7.45E-9	day of month, month, RSI(22), RSIS(10), ROC(11), ROC(40)	% Change
	PaceRegression	-E pace4	% price change, month, LAG(1), LAG(8), RSIS(10), ROC(31)	% Change
	RBFNetwork	-B 6 -S 6 -R 5.44E-8 -M -1 -W 0.28	price direction, close price, day of month, LAG(8), WILRS(35)	Class
	IB1		close price, RSIS(4)	Class
	IBk	-K 38 -W 0 -E -I -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	month, WILR(20), RSIS(12), ROC(14)	% Change
	SimpleCart	-S 6 -M 2 -N 6 -U -H -C 0.64	close price, WILR(4), RSIS(12)	Class
	ConjunctiveRule	-N 2 -M 7.93 -P -1 -S 6	price direction, close price, day of month, month, LAG(6)	Class
	M5Rules	-U -M 1	% price change, RSI(16), RSIS(7), ROC(19), ROC(40)	% Change
	PART	-M 4 -C 0.44 -Q 2	price direction, day of week (nom), month, LAG(8), RSI(6), RSI(10), ROC(15)	Class
	Ridor	-F 2 -S 1 -N 7 -A -M	% price change, day of week (nom), day of month, month, LAG(7)	Class
IBM	LeastMedSq	-S 10 -G 2	price direction, day of month, MA(30), WILR(33), ROC(6), ROC(27)	% Change
	Logistic	-R 0.09 -M -1	close price, day of week (num), WILR(10), WILR(20)	Class
	PaceRegression	-E olsc -S 2.0	close price, month, day of week (num), WILR(18), ROC(9)	% Change
	VotedPerceptron	-I 3 -E 2.98 -S 9 -M 19838	close price, day of month, LAG(3), LAG(7), WILR(14), WILR(31), RSI(36)	Class
	IBk	-K 34 -W 8 -I -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	LAG(4), WILR(11), RSI(37), ROC(7), ROC(19)	Class
	KStar	-B 41 -M d	MA(7), MA(26), WILR(5), WILRS(17), ROC(18), ROC(33)	Class
	ADTree	-B 9 -E -1 -D	WILRS(11), RSI(6), RSI(13), ROC(16)	Class

	BFTree	-S 2 -M 8 -N 8 -C 1.0 -P POSTPRUNED	day of week (nom), month, WILRS(5), WILRS(19), WILRS(24), RSIS(36), ROC(24)	Class
	REPTree	-M 3 -V 0.001 -N 5 -S 9 -L -1	MA(39), RSIS(32), ROC(18)	Class
	ConjunctiveRule	-N 5 -M 3.43 -P 2 -S 4	close price, day of week (nom), day of month, WILRS(9), WILRS(28), RSIS(3), RSIS(37)	% Change
	Ridor	-F 6 -S 1 -N 5 -A -M	RSI(5), RSI(24), RSI(31), RSIS(7)	Class
INTC	LibSVM	-S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1	LAG(7), WILR(14), RSIS(22), ROC(11)	Class
	VotedPerceptron	-I 1 -E 3.11 -S 16 -M 9886	close price, day of month, MA(34), WILR(27), WILRS(30), ROC(35)	Class
	KStar	-B 28 -M m	day of month, LAG(4), RSI(15)	% Change
	ADTree	-B 19 -E -2 -D	close price, day of week (nom), day of month, MA(10), WILRS(36), RSIS(21)	Class
	J48	-L -S -R -N 3 -Q 1 -M 18	close price, day of month, month, MA(21), WILRS(9)	Class
	BFTree	-S 6 -M 20 -N 6 -G -C 0.77 -P POSTPRUNED	price direction, day of month, month, day of week (num), WILR(27)	Class
	IBk	-K 2 -W 13 -E -I -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	close price, day of week (nom), month, LAG(7), WILRS(12), RSIS(18)	Class
	REPTree	-M 7 -V 0.001 -N 3 -S 10 -L 29	close price, day of month, month, day of week (num), WILRS(5), WILRS(7), RSI(36)	Class
	IB1		% price change, month, RSIS(28)	Class
	JRip	-F 5 -N 8.62 -O 4 -S 6	price direction, close price, day of week (nom), day of month, month, RSIS(23), ROC(15)	Class
	M5Rules	-U -M 3	close price, month, MA(17), LAG(1), WILR(9), RSI(18)	% Change
JNJ	PaceRegression	-E nested	day of month, day of week (num), MA(11), MA(26), MA(36), WILRS(9), RSIS(31)	% Change
	RBFNetwork	-B 5 -S 7 -R 1.9E-8 -M -1 -W 0.24	close price, day of month, RSIS(34)	Class
	IB1		price direction, close price, day of week (num), RSI(34), ROC(17)	Class
	FLR	-R 0.58 -Y -B	close price, day of month, LAG(1), WILR(12), WILR(32), ROC(36)	Class
	M5P	-U -M 35	% price change, close price, day of week (num), MA(16), MA(17), WILR(26), RSIS(26)	% Change
	REPTree	-M 11 -V 0.001 -N 4 -S 8 -L -1	% price change, close price, day of week (num)	Class
	SimpleCart	-S 6 -M 2 -N 4 -U -C 0.74	price direction, close price, day of week (nom), month, MA(38), WILR(3), WILR(21), RSI(8)	Class
	Ridor	-F 2 -S 1 -N 6	price direction, close price, RSIS(10), ROC(18)	Class
	MultilayerPerceptron	-L 0.37 -M 0.6 -N 481 -V 0 -S 10 -E 20 -H t -C	% price change, day of week (nom), month, MA(3), WILRS(17), RSIS(3)	Class
	SimilarityClassifier	-S 0.32 -W correlation -C unsupervised	% price change, close price, day of week (nom), MA(4), MA(19), RSI(11), RSI(36)	Class
	DistanceToAverage	% price change, close price, day of month, LAG(7), WILRS(14), RSI(4), RSI(14), RSIS(31)	Class	
KFT	NaiveBayes	-K	% price change, month, WILRS(24), RSI(8), RSIS(15)	Class
	LinearRegression	-S 0 -R 3.1E-8	price direction, day of week (nom), LAG(5), LAG(8), WILRS(15), RSI(10), ROC(2)	% Change
	Logistic	-R 0.071 -M -1	day of month, month, WILR(10), WILR(21), WILRS(26), RSI(15), RSI(28), ROC(1)	Class

	PaceRegression	-E eb	day of week (num), MA(15), WILRS(40)	% Change
	IB1		price direction, day of month, month, LAG(2), WILR(40), WILRS(37), RSIS(2)	Class
	KStar	-B 30 -M a	price direction, month, ROC(30)	Class
	BFTree	-S 4 -M 2 -N 8 -H -C 0.55 -P POSTPRUNED	price direction, close price, day of month, MA(13), WILR(31), WILRS(12), ROC(2)	Class
	LMT	-B -C -I -1 -M 17 -W 0.523 -A	price direction, day of month, day of week (num), MA(21), WILRS(25), WILRS(26), RSIS(32), ROC(29)	Class
	NBTree		price direction, close price, day of week (nom), day of month, month, MA(40), WILRS(38), RSIS(15)	Class
	REPTree	-M 2 -V 0.001 -N 2 -S 6 -L 28	% price change, day of week (nom), day of month, month, LAG(7), RSI(8), ROC(4)	Class
	JRip	-F 5 -N 4.56 -O 4 -S 8	% price change, day of week (nom), WILRS(24), RSIS(18)	Class
KO	JRip	-F 3 -N 6.6 -O 5 -S 9	% price change, close price, day of month, month, day of week (num), MA(24), WILR(13)	Class
	LeastMedSq	-S 4 -G 4	% price change, day of month, month, LAG(4), WILR(34), RSI(6)	% Change
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -H -Z	% price change, day of month, month, day of week (num), LAG(3), LAG(5), WILRS(40), RSIS(26)	% Change
	LinearRegression	-S 1 -C -R 4.75E-8	price direction, day of month, day of week (num), MA(16), WILR(6), WILRS(8), RSI(7), ROC(2)	% Change
	PaceRegression	-E pace6	day of month, month, MA(10), LAG(3), LAG(4), WILR(20), ROC(5)	% Change
	BFTree	-S 6 -M 11 -N 7 -R -C 1.0 -P POSTPRUNED	% price change, close price, day of week (nom), month, MA(9), MA(12), WILRS(2), RSIS(26)	Class
	IB1		day of month, month, WILR(16), WILR(26), WILRS(39), WILRS(40), RSI(6)	Class
	M5P	-U -M 8 -L	day of week (num), MA(14), RSI(2), RSI(18)	% Change
	Logistic	-R 0.07 -M 6	price direction, close price, day of month, day of week (num), MA(10), WILR(25), RSI(16)	Class
	M5Rules	-M 14	% price change, day of week (nom), MA(6), LAG(8), WILRS(14), RSI(2), RSIS(8)	% Change
	IBk	-K 35 -W 8 -F -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	month, WILRS(40), RSI(12), RSI(16), RSI(32), RSIS(23), ROC(6)	Class
MCD	NaiveBayes	-K	day of month, WILR(23), WILR(25), WILRS(16), RSI(6), ROC(3)	Class
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -B	day of month, month, MA(10)	% Change
	LinearRegression	-S 1 -C -R 8.38E-8	day of month, month, day of week (num), MA(9), LAG(8), RSI(33), RSIS(37), ROC(18)	% Change
	Logistic	-R 0.063 -M 3	% price change, day of month, day of week (num), MA(31), LAG(5), WILRS(33), RSI(40)	Class
	PaceRegression	-E pace2	month, MA(4), MA(30), RSIS(15)	% Change
	ADTree	-B 14 -E 89 -D	% price change, day of week (nom), day of month, month, MA(23), MA(28), RSIS(35), ROC(22)	Class
	BFTree	-S 7 -M 9 -N 4 -H -G -C 1.0 -P POSTPRUNED	WILRS(15), RSI(26), ROC(6)	Class

	LMT	-B -P -I 30 -M 30 -W 0.62	price direction, day of month, day of week (num), MA(22), MA(40), RSIS(29), ROC(3)	Class
	M5P	-U -M 44 -L	% price change, day of week (nom), day of month, month, WILRS(23), RSIS(15)	% Change
	M5Rules	-M 4	price direction, day of month, month, day of week (num), MA(10), LAG(3), LAG(4), WILRS(37)	% Change
	NNge	-G 5 -I 3	day of week (nom), LAG(1), RSI(5)	Class
MRK	Logistic	-R 0.015 -M -1	close price, day of month, day of week (num), MA(10), LAG(1), WILR(18), RSIS(13), ROC(24)	Class
	VotedPerceptron	-I 3 -E 1.645 -S 9 -M 9583	price direction, day of month, month, day of week (num), LAG(1), WILR(40)	Class
	IB1		% price change, day of week (nom), day of month, month, WILRS(38)	Class
	KStar	-B 26 -M m	close price, day of month, month, WILRS(17)	Class
	VFI	-B 0.867	close price, month, WILR(21), WILRS(39), ROC(1)	Class
	ADTree	-B 14 -E 61	% price change, close price, day of week (nom), LAG(1), RSI(2), RSIS(20), RSIS(34)	Class
	J48	-L -S -C 0.43 -M 9	close price, day of week (nom), day of month, month, LAG(7), WILRS(33)	Class
	REPTree	-M 2 -V 6.482E-4 -N 2 -S 3 -L 26	price direction, close price, day of month, day of week (num), RSI(30), RSIS(14)	Class
	SimpleCart	-S 4 -M 1 -N 4 -C 1.0	close price, RSI(31)	Class
	JRip	-F 5 -N 1.2 -O 2 -S 3 -E	day of month, month, day of week (num), RSIS(20), ROC(25)	Class
	M5Rules	-M 5	close price, MA(5), MA(6), MA(11), WILR(37)	% Change
MSFT	NaiveBayes	-K	% price change, close price, day of month, month, MA(19), WILRS(18), RSIS(35), ROC(11)	Class
	LeastMedSq	-S 4 -G 7	price direction, day of week (num), MA(10), RSI(36), RSIS(9)	% Change
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -H -B	% price change, day of week (num), MA(19), WILR(9), WILR(12), WILR(33), ROC(14)	% Change
	PaceRegression	-E subset	% price change, close price, WILRS(30), RSI(15)	% Change
	VFI	-B 0.504	% price change, MA(23), WILR(29), RSI(30), RSIS(16), RSIS(26), ROC(2)	Class
	PART	-B -M 6 -C 0.33 -Q 7	price direction, close price, day of month, WILR(4), WILR(13), WILRS(7), RSI(21)	Class
	KStar	-B 45 -M m	price direction, close price, day of week (nom), LAG(4), LAG(7), WILR(25), RSI(37), ROC(3)	Class
	SimpleCart	-S 4 -M 3 -N 6 -H -C 0.502	price direction, day of month, day of week (num), MA(8), WILR(4), WILR(18), WILR(31), RSI(10)	Class
	REPTree	-M 13 -V 0.002 -N 6 -S 7 -L -1	% price change, day of month, MA(36), WILRS(25), RSIS(14), RSIS(31), ROC(4)	Class
	JRip	-F 4 -N 3.86 -O 4 -S 5 -E	day of month, month, day of week (num), LAG(5), WILR(3), ROC(18), ROC(24), ROC(30)	Class
	ConjunctiveRule	-N 2 -M 7.92 -P -1 -S 3	price direction, close price, month, day of week (num), LAG(2), WILRS(6), RSIS(19), ROC(35)	Class
NVDA	LeastMedSq	-S 7 -G 1	price direction, close price, day of week (nom), WILR(33), WILR(37), ROC(8), ROC(25), ROC(38)	% Change

	Logistic	-R 0.013 -M -1	% price change, day of month, WILR(31), WILRS(21), ROC(38)	Class
	VotedPerceptron	-I 1 -E 4.29 -S 19 -M 1110	price direction, day of month, WILR(9), WILR(36)	Class
	PaceRegression	-E aic	% price change, close price, day of week (num), MA(16), WILRS(3), WILRS(11), RSIS(38)	% Change
	Ridor	-F 5 -S 1 -N 4	% price change, MA(16), RSIS(40)	Class
	JRip	-F 3 -N 2.29 -O 5 -S 2	month, day of week (num), WILR(3), WILR(26), RSIS(14), ROC(9)	Class
	REPTree	-M 2 -V 5.47E-4 -N 6 -S 4 -L 17	price direction, month, day of week (num), LAG(3), ROC(5), ROC(29)	Class
	ConjunctiveRule	-N 3 -M 6.86 -P -1 -S 3	close price, day of month, day of week (num), LAG(2), RSI(2), RSIS(20), ROC(27)	Class
	M5Rules	-M 5	% price change, day of month, day of week (num), LAG(7), RSI(9), RSI(22), RSI(34), ROC(4)	% Change
	SimpleCart	-S 2 -M 5 -N 4 -U -H -C 0.73	close price, day of week (nom), month, MA(18), WILRS(32), WILRS(34), ROC(34)	Class
	LMT	-P -I 36 -M 8 -W 0.81	price direction, month, day of week (num), WILRS(15), WILRS(24)	Class
	NaiveBayes		price direction, month, MA(29), MA(35), LAG(8), RSI(31), RSIS(39)	Class
	Logistic	-R 0.04 -M 14	price direction, close price, day of week (num), MA(5), WILR(11), WILRS(38), ROC(4)	Class
	RBFNetwork	-B 4 -S 7 -R 2.25E-8 -M -1 -W 0.14	% price change, day of week (nom), month, MA(2), WILRS(25), RSI(19), RSIS(27), ROC(32)	% Change
	VotedPerceptron	-I 2 -E 0.75 -S 2 -M 7723	% price change, day of week (nom), MA(14), LAG(5), LAG(7), WILR(12)	Class
	KStar	-B 15 -M n	price direction, day of month, month, day of week (num), LAG(3), WILR(28)	Class
PFE	FLR	-R 0.59 -B	day of month, MA(39), RSI(21)	Class
	VFI	-B 0.84	price direction, close price, day of month, MA(11), WILRS(35), RSI(18), RSIS(7)	Class
	J48	-C 0.39 -B -M 4 -A	price direction, day of month, day of week (num), MA(13), WILR(3), WILR(39), ROC(27)	Class
	SimpleCart	-S 1 -M 5 -N 3 -U -C 0.7	% price change, close price, day of week (nom), month, MA(20), WILRS(24), RSI(32), RSIS(28)	Class
	ConjunctiveRule	-N 5 -M 2.59 -P -1 -S 8	% price change, close price, day of week (num), RSI(25)	Class
	PART	-M 6 -C 0.16 -Q 1	price direction, close price, day of month, month, day of week (num), WILRS(3)	Class
	LeastMedSq	-S 10 -G 4	% price change, day of week (nom), MA(22)	% Change
	LibSVM	-S 3 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40 -C 1.0 -E 0.001 -P 0.1 -Z	% price change, close price, MA(29), WILRS(7), RSI(11)	% Change
	IB1		price direction, WILRS(11), WILRS(37)	Class
T	IBk	-K 30 -W 11 -E -F -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	price direction, day of week (nom), month	Class
	KStar	-B 17 -M a	day of week (num), WILR(13), ROC(5)	Class
	BFTree	-S 3 -M 2 -N 4 -C 1.0 -P PREPRUNED	% price change, close price, day of week (num), MA(6), MA(19), MA(29), WILRS(33)	Class
	J48	-C 0.355 -B -M 5	day of week (nom), month, WILR(26), RSIS(8)	Class

	SimpleCart	-S 10 -M 4 -N 3 -U -H -C 1.0	close price, month, day of week (num), LAG(2), WILR(37), WILRS(40), RSI(7)	Class
	M5Rules	-U -M 10	% price change, day of week (nom), MA(23), RSI(2), RSI(30), RSIS(27)	% Change
	NNge	-G 7 -I 4	day of week (nom), day of month, LAG(7), WILR(36)	Class
	PART	-M 7 -C 0.152 -Q 4	% price change, close price, day of month, month, MA(36), RSIS(15)	Class
VZ	Logistic	-R 6.44E-4 -M -1	price direction, close price, LAG(5), LAG(8), WILRS(24), RSI(39), RSIS(23)	Class
	VotedPerceptron	-I 3 -E 2.37 -S 18 -M 1259	% price change, day of month, day of week (num), MA(27), LAG(5), WILR(5), RSI(14), RSIS(32)	Class
	IB1		MA(25), LAG(5), WILRS(37), RSI(34)	Class
	IBk	-K 49 -W 10 -E -I -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	price direction, day of month, month, WILR(20), WILR(38), WILRS(12)	Class
	KStar	-B 2 -M n	day of week (nom), MA(34), ROC(28)	Class
	ADTree	-B 18 -E 7	% price change, close price, day of month, day of week (num), WILR(35), WILRS(8), WILRS(23), RSIS(33)	Class
	J48	-S -C 0.48 -M 39	price direction, close price, day of week (num), LAG(7), WILR(20), WILRS(28), ROC(8)	Class
	M5P	-M 6	close price, MA(18), WILR(9), WILRS(39), RSI(23), ROC(38)	% Change
	NBTree		price direction, close price, day of month, MA(14), LAG(3), WILRS(14), RSI(10), ROC(28)	Class
	REPTree	-M 2 -V 0.001 -N 5 -S 5 -L 24	% price change, close price, day of week (num), MA(2), MA(7), WILRS(6), WILRS(14), RSI(5)	Class
	PART	-B -M 10 -C 0.23 -Q 7	% price change, close price, WILR(8)	Class
WMT	LeastMedSq	-S 8 -G 4	day of week (nom), day of month, WILR(16)	% Change
	IB1		% price change, close price, day of week (nom), MA(15), LAG(4), RSIS(9), RSIS(14), RSIS(26)	Class
	IBk	-K 1 -W 12 -E -F -A "weka.core.neighboursearch.LinearNNSearch -A |weka.core.EuclideanDistance -R first-last|"	day of month, RSIS(32), RSIS(37)	Class
	ADTree	-B 16 -E -2	close price, day of week (num), WILR(16), RSI(24), RSIS(6), ROC(7)	Class
	M5P	-M 43 -L	% price change, close price, day of month, day of week (num), MA(7), WILR(20), RSI(22), RSIS(15)	% Change
	REPTree	-M 2 -V 0.001 -N 6 -S 10 -L -1	close price, WILRS(18), RSI(5)	Class
	SimpleCart	-S 3 -M 3 -N 6 -U -H -C 1.0	close price, day of month, ROC(5)	Class
	ConjunctiveRule	-N 5 -M 3.12 -P 3 -S 3	price direction, day of month, RSIS(35)	% Change
	JRip	-F 5 -N 9.39 -O 1 -S 7	price direction, close price, day of month, RSIS(25), ROC(9)	Class
	M5Rules	-U -M 14	% price change, day of month, month, MA(26), WILR(2), WILRS(22), WILRS(32), ROC(7)	% Change
	Ridor	-F 5 -S 1 -N 10 -A	% price change, close price, day of month, WILRS(11), WILRS(31), RSI(14), RSI(34), RSIS(23)	Class
XOM	LinearRegression	-S 1 -C -R 5.34E-8	month, WILR(28), RSI(30), ROC(1)	% Change
	Logistic	-R 0.1 -M 5	day of week (nom), day of month, WILR(27), ROC(9), ROC(19)	Class
	RBFNetwork	-B 4 -S 1 -R 5.58E-8 -M -1 -W 0.09	month, WILR(16), WILR(30), WILRS(14), RSIS(35), ROC(5), ROC(19)	% Change

	KStar	-B 4 -M a	close price, day of week (nom), day of month, month, LAG(4), WILRS(31), RSI(17)	Class
	ADTree	-B 18 -E -1 -D	close price, day of week (nom), WILR(25), RSIS(6), ROC(17)	Class
	BFTree	-S 5 -M 10 -N 7 -G -A -C 1.0 -P PREPRUNED	price direction, day of month, day of week (num), MA(7), LAG(4), WILRS(25)	Class
	J48	-S -R -N 3 -Q 1 -B -M 27	price direction, day of week (nom), WILR(24), WILR(35)	Class
	ConjunctiveRule	-N 3 -M 5.55 -P -1 -S 7	close price, month, WILRS(15), ROC(26)	% Change
	DecisionTable	-X 5 -I -R -S "weka.attributeSelection.RandomSearch -F 25.0"	price direction, close price, day of week (nom), day of month, month, MA(17), MA(22), WILR(35)	Class
	PART	-B -M 10 -C 0.21 -Q 4	close price, day of week (nom), day of month, month, LAG(5), WILR(40), WILRS(9), RSIS(10)	Class
	Ridor	-F 2 -S 1 -N 5 -A	price direction, close price, day of month, month, MA(12), WILR(32), WILRS(9), ROC(33)	Class