Forecasting – where computational intelligence meets the stock market

Edward Tsang

Working Paper WP026-08
Centre for Computational Finance and Economic Agents (CCFEA)
University of Essex, UK
Revised December 2008

Abstract

Forecasting is an important activity in finance. Traditionally, forecasting has been done with in-depth knowledge in finance and the market. Advances in computational intelligence have created opportunities that were never there before. Computational finance techniques, machine learning in particular, can dramatically enhance our ability to forecast. They can help us to forecast ahead of our competitors and pick out scarce opportunities. This paper explains some of the opportunities offered by computational intelligence and some of the achievements so far. It also explains the underlying technologies and explores the research horizon.

1. Beating the market

Wouldn't it be nice if you can tell us whether stock prices will go up or down tomorrow? Numerous attempts have been made to forecast stock prices. Motivation is not limited to financial gains. Financial stability could be improved should regulators be able to recognize patterns that signal market failure.

Traditional approaches to forecasting can roughly be divided into two paradigms: *fundamental analysis* and *technical analysis*. Fundamental analysts attempt to study the "true value" of financial assets, such as shares. If their market prices deviate from the true value perceived, prices are expected to return to the true value some time in the future. If the deviation is large enough, it signals an opportunity to buy or sell. Technical analysts attempt to observe the movement of the prices, and discover patterns in it. If patterns can be found, then they will be used to recognize investment opportunities.

The key question that fundamental analysts face is how to assess the value of an asset. What information is relevant to the calculation? Costs are involved in acquiring and processing such information. Research and computation involve expertise, expenses and time. Could computational intelligence be used to boost expertise and reduce research and computation cost and time? *Optimization* is an important branch of computational intelligence. Could it help?

Technical analysts also face a number of key questions. What information is relevant to forecasting the market? Do patterns exist? Even if they do, could one find them? Even if one manages to find patterns, would those patterns repeat themselves in the future? *Machine learning* is an important branch of computation intelligence. Could it help finding patterns?

2. The bad news: the efficient market hypothesis

The bad news is: classical economics tell us that the market cannot be predicted. Fama [3] defined the concept of *efficient capital markets*. What it says is that if a market is "efficient", prices will always fully reflect the available information that relates to the financial asset being traded. Therefore, it is impossible to consistently outperform the market (i.e. earning more than what the market as a whole offers – often referred to as *excess return*) by using information that is available on the market. This is known as the *efficient market hypothesis* (EMH).

Classical economists believe that markets are by and large efficient, though efficiency can be interpreted in different ways. The weak-form of EMH suggests that one cannot gain excess return based on past share prices and returns. The semi-strong-form of EMH suggests that one cannot gain excess return based on information that is publicly available; new information (such as company announcements and change of interest rates) will quickly be reflected in the price. The strong-form of EMH suggests that one cannot gain excess return even if one has insider information.

If EMH holds (at least in semi-strong-form), any forecasting attempt is futile. The price of an asset always reflects its true value, which is the result of fundamental analysis, and therefore does not reflect any potential excess return in the future. Technical analysis is futile too. This is because patterns, even if they exist and repeat themselves, are consequences of price movements, which reflect the dissemination of new information. They therefore have no predictive power.

3. Is the market really "efficient"?

Many have examined the EMH, both theoretically and empirically. Empirical results have been found, both in support of the EMH and its contrary. In general, early research provided strong evidence in support of weak-form efficiency. In contrast, recent works tend to illustrate anomalies, which pose challenges to the weak-form of EMH [20]. As always, statistics tend to support results that one is looking for.

EMH also faces a lot of challenges from the theoretical front. EMH is built upon a number of assumptions [3]. Some of the fundamental assumptions include:

a) Frictionless information flow: information item is equally and instantaneously available to all market participants.
b) Perfect rationality: all participants are fully rational in decision making.
c) Homogeneous expectations: all participants agree on the implications of each information item on the price of each security.
d) Costless analysis: analysis of each information item is costless.

Some might be prepared to believe that information flow (especially price and return) is close to frictionless (assumption (a)), but full rationality and homogeneous expectations (assumptions (b) and (c)) attract a lot of debate. Coming from a computational angle, we know that analysis is not costless (assumption (d)). If it is, then computer science is not a worthy subject. This is an important point, which needs a bit of elaboration.

Despite the speed of modern computers, many problems are still beyond our reach. This is due to the fact that many problems involve the consideration of an exponential number of combinations – a phenomenon known as the *combinatorial explosion* problem. For example, in chess, all the rules are clearly defined. Therefore, in principle, all possible moves can be evaluated exhaustively. However, as one increases the number of moves that one looks ahead, the combinations of moves grow exponentially. For example, suppose for simplicity that in every move, a player has 20 options (obviously this number varies as the game develops – increasing at the opening and decreasing towards end-game, but that does not fundamentally affect our discussion here). If one looks ahead for 30 moves in a brute-force manner, there are 20 to the power 30 (i.e. in the order of ten to the power 39) possible board positions to evaluate. Even the fastest computers today will take millions of years (in fact much more) to evaluate them. Much of computer science involves the studying of how to compute quickly; i.e. to do better than brute-force search.

The market is only efficient if enough investors have the capability of analysing the true values of assets, and assessing the impact of new information. Not everyone has the same computational competence. Perhaps optimization, machine learning and other computational intelligence techniques still have some roles to play.

4. What would be a reasonable research agenda?

Could computational intelligence help us to create a crystal ball that tells us the exact price of a stock tomorrow? That would be very useful indeed. However, knowing the exact price tomorrow is probably too much to ask for. I am happy to settle for less.

Suppose I know that a coin is biased. Suppose I know that it has a 60% of turning up with head with it is flipped. Equipped with this piece of information, I manage to turn a 50%-50% game to 60%-40% in my advantage. This piece of information does not necessary guarantee winning in every game. Not only does it guarantee winning in the long run. But it increases my chance of winning. That may be good enough.

What could we aim to achieve in forecasting research? We could attempt to predict prices in the future. Technically speaking, this task is to find a function that maps input values to a numerical value. Input to this function could be past prices, returns, dividends, or any values that the analyst may want to use, including fundamental values derived from the analyst's model. In fact, the practical difficulty is to know what inputs might be relevant, given that one could look at any indicators in the company or the economy, and derive any number of ratios from the other inputs.

Instead of attempting to predict future prices, one could also attempt to answer questions of the following form:

"*Will the price go up (or down) by at least r% within the next n days?*"

If one could answer this question, one could identify investment opportunities. Answering this question is arguably easier than predicting future prices. The task here is to find a Boolean function that maps the input values to a "yes" or "no" value.

It is worth stating the obvious that we do not have to find all situations under which price will go up (or down) by at least *r*% within *n* days. In other words, we do not have to find *all* investment opportunities. We could also attempt to answer the opportunity question with different values for *r* (target return) and *n* (investment horizon). Besides, we could attempt to forecast for different stocks. Doing all these will not only potentially increase the number of opportunities being detected, but also help to build a bigger picture of the market.

5. How can computational intelligence help?

There is no guarantee that functions for price prediction or investment opportunity detection exist. However, if they do, then the next question is "how to fund them?" Human expertise is essential. In this paper, I would like to argue that machine learning is a discipline that one could not ignore.

Suppose you want to answer the question "will Company X's price rise by at least 6% within the next 21 days?" Suppose, through years of research, you have discovered that a certain ratio *R* is a good indicator on the movement of the share price of Company X. Suppose you are right. With such insight, you expect yourself to be able to out-smart other investors who have discovered *R*. However, you don't know exactly how share prices will move in response to changes in *R*. To use the terms introduced in the previous section, you have not discovered any function that maps *R*, together with other market indicators, to investment opportunities. What can you do?

For argument's sake, suppose it is a fact that whenever *R* has a value outside 1.4 and 2.7, the volatility of the share prices is above 2.5, and the share's yield is above 5.7%, then prices will rise by at least 6% within the next 21 days. The question is how would you find this fact? The number of market indicators is vast. Of all the market indicators available, how could you discover that the volatility and yield interact with *R*? How could you discover values such as 1.4, 2.7, 2.5 and 5.7%?

We explained in the previous section that the task on hand is a function fitting problem: one attempts to find a Boolean function using inputs which include *R*, volatility, yield and any other indicators. The hypothetical function mentioned above could be expressed in a Boolean function of the following form:

```
IF ((($R$ < 1.4 OR $R$ > 2.7) AND Volatility > 2.5) AND Yield > 5.7)
THEN TRUE
ELSE FALSE
```

False here is best interpreted (in line with logic programming) as "*no, this rule does not suggest that r% can be achieved within the next n days*" rather than "*no, price will not rise by r% in n days*". In other words, while this rule does not suggest that the target price will be reached within n days, it does not contradict with other rules that may say so.

6. Machine learning, an introduction

In the previous section, we have established that forecasting can be seen as a task of finding functions that may input values numerical or Boolean values. In the rest of this paper, we shall focus on Boolean functions.

Function fitting is an application that many machine learning methods have been applied to. These methods search in the space of functions permitted by a given grammar. For the purpose of classification, the fitness of a function is judged by its correctness in classifying the given data. The components and structure of the function and the thresholds (1.4, 2.7, etc.) mentioned in the previous section are subjects of the search.

Machine learning typically involves finding patterns (functions in our context) that fit the data available to the learner, and hope that such patterns apply to unseen data. A data set typically contains many instances of observations. Table 1 shows an example of ten instances in a data set. This is a data set that could have supported the hypothetical function above. Each instance comprises *R*, Volatility and Yield and possibly other variables. Column (e) is the target, which is available to the researcher in hindsight. For example, Instance 1 shows that *R* is 1.2, volatility is 3.1 and yield is 4.8. If these values are fed into the Boolean function above, the function will return "false" because the condition "yield > 5.7" is false.

Table 1: Example of a partial data set for training in machine learning

(a) Instances	(b) *R*	(c) Volatility	(d) Yield	…	(e) Target
1	1.2	3.1	4.8		False
2	1.3	3.0	6.6		True
3	2.8	2.9	5.9		True
4	2.5	1.7	7.0		False
5	2.4	3.5	6.9		False
6	2.0	2.9	5.6		False
7	3.1	3.3	5.8		True
8	3.1	3.0	5.5		False
9	2.8	2.4	5.0		False
10	2.6	2.5	5.2		False
….					

In machine learning, information obtained in hindsight is used for training a system. Functions found by the system are typically tested on unseen data (i.e. data that was not used for training). Each function that the system generates classifies the instances into true or false – true meaning that in that particular instance the target return was achievable within the time horizon; false means otherwise. The challenge is to find functions which outputs match all the values in the "Target" column.

Table 2 shows the classifications of the data in table 1 by a hypothetical function F, content of which is unimportant here. The attention here is about how the fitness of a function is evaluated.

Table 2: Example of classifications in a sample data set by a hypothetical function F

(a) Instances	(b) *R*	(c) Volatility	(d) Yield	(e) Target	(f) Classifications
1	1.2	3.1	4.8	False	False
2	1.3	3.0	6.6	True	True
3	2.8	2.9	5.9	True	False
4	2.5	1.7	7.0	False	False
5	2.4	3.5	6.9	False	False
6	2.0	2.9	5.6	False	False
7	3.1	3.3	5.8	True	True
8	3.1	3.0	5.5	False	True
9	2.8	2.4	5.0	False	True
10	2.6	2.5	5.2	False	False

By comparing columns (e) and (f) in Table 2, one can see that the function F has made some correct classifications, and some misclassifications. Five of the False instances were correctly classified, but two False instances were misclassified as True (namely Instances 8 and 9). Two of the True instances were correctly classified, but one True instance was misclassified as False (namely Instance 3). The performance of function F can be summarized in a *confusion matrix*, as shown in Table 3.

Table 3: Confusion matrix of the performance of function F in Table 2

	False in Classification	True in Classification	Total
False in reality	True Negative: 5	False Negative: 2	6
True in reality	False Positive: 1	True Positive: 2	4
	7	3	10

It would be useful to introduce some performance measures here. The overall rate of *correctness* in a classification can be measured by the total number of correct classifications divided by the total number of instances. The *precision* of a classification is the number of correctly classified True instances divided by the total number of instances classified as True. The recall is the number of True instances in reality that were classified as True. The performances of function F as shown in Table 3 are therefore as follows:

Correctness = correct classifications ÷ Total number of instances = (5 + 2) ÷ 10 = 70%

Precision = True Positive ÷ Number of instances classified as True = 2 ÷ 4 = 50%

Recall = True Positive ÷ Number of True instances in reality = 2 ÷ 3 = 66.7%

Depending on the application, some may prefer to have high precision, and some may prefer to have high recall. For example, in finding investment opportunities, we may weigh precision higher than recall. This is because if a False instance is misclassified as True, the target return may not be achievable; it may even lead to a loss. For preliminary scanning in medical applications, False alarm will cause anxiety and further examinations, but that is preferred to missing life-threatening diseases. Therefore, one may put more weigh on recall than precision.

7. Genetic Programming in Forecasting

Evolutionary computation [7] is a branch of computer science. It borrows Darwin’s idea of evolution, where survival depends on fitness. In the context of function fitting, the idea is to maintain a set of candidate functions. Each function is evaluated for its fitness, which is measured in terms of their correctness in classifying the data. The functions are then modified in an artificial environment that mimics evolution, which eliminates weak individuals and allow fitter individuals to pass their genetic material to later generations.

Functions are often represented by trees in computer science. Since we do not know in advance what sorts of trees make good strategies, we need a representation that allows us to generate trees of any structure any size. Tree is a dynamic data structure, which means it can be used to represent functions of any size (within practical limit). *Genetic programming* is a branch of evolutionary computation that evolves trees [10][11].

Therefore, it is a good candidate for machine learning for finding investment opportunities.

An early publication that reported the application of genetic programming to financial forecasting was by Neely et al [16]. They showed that technical rules can be generated by genetic algorithms to beat the foreign exchange market. Their rules gained excess returns on major currency exchanges without incurring extra risk. This is significant because the results suggested that patterns can be found in historical prices and those patterns appeared to repeat themselves. Moreover, this work gave evidence that genetic programming can be used to find such patterns.

EDDIE (which stands for Evolutionary Dynamic Data Investment Evaluator) is an umbrella under which a number of programs have been developed. James Butler developed the first version of EDDIE, which was used for forecasting in horse racing [26]. Jin Li and the author later enhanced its effectiveness and applied it to financial forecasting [13]. Then it was developed into an interactive tool for finding investment opportunities [28]. EDDIE uses genetic programming to learn functions that identify investment opportunities.

Functions are represented by *genetic decision trees* in EDDIE. Within the tree representation, there are many ways to represent investment decisions. Figure 1 shows an example of a genetic decision tree that contains the rule shown in Section 5. The root of the tree represents an IF-THEN-ELSE function, which takes three arguments. The first argument, shown as the left branch in Figure 1, represents a condition. If the condition is met, then the decision will be determined by the second argument, which is True in the tree shown in Figure 1. When the condition is not met, the decision will be determined by the third argument, which is a function represented by the sub-tree on the right-most branch in Figure 1.

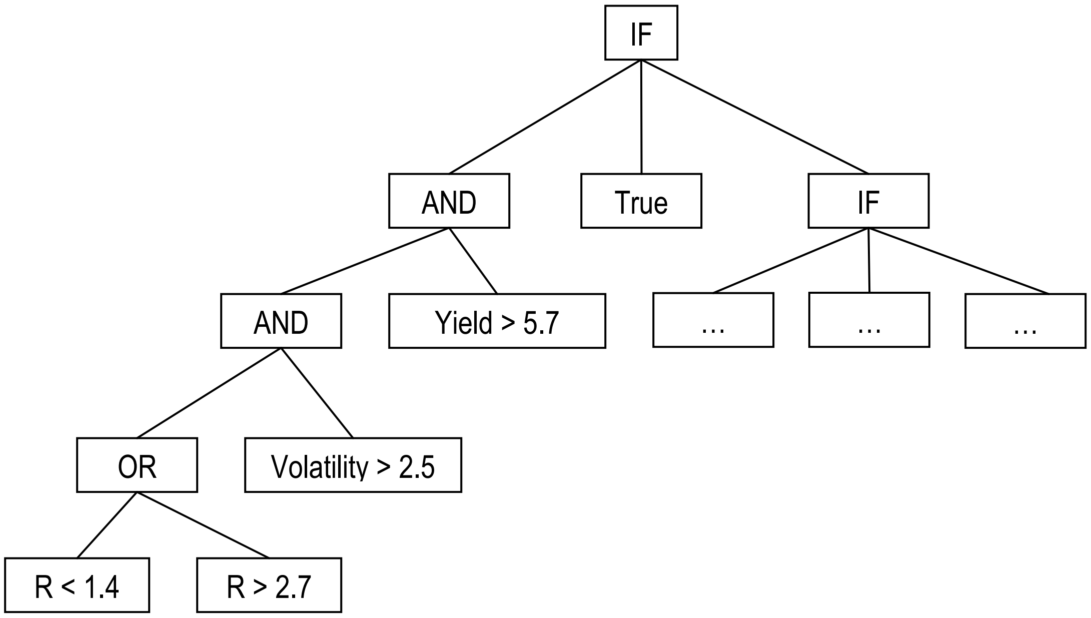

Figure 1: Example of a Genetic Decision Tree (GDT)

EDDIE has been tested extensively, on a large number of data sets. Even when limited to using basic textbook indicators (e.g. see [2][24]) EDDIE was demonstrated to find technical trading rules in the stock market [12].

8. Inside the box – how does EDDIE work?

Here we shall briefly explain how EDDIE works. Readers interested in the technical details are referred to [27]. Given a set of variables, EDDIE attempts to find interactions among them. Following the practice of genetic programming, EDDIE generates a population of (e.g. 1,000) random functions. Then it iterates in a predefined number of evolutionary generations (e.g. 50). In each generation, two parents are picked. EDDIE picks each parent by a "tournament", which means selecting the fittest individual from a number of (e.g. 4) randomly-picked individuals from the population. The two parents generate two offspring by exchanging part of their genetic building blocks. In this case, each parent is a tree, and their genetic building blocks are branches (i.e. sub-trees). The offspring thus generated form the new population.

A bit of explanation on the principle of EDDIE, or evolutionary computation in general is necessary for the understanding of the rest of the article. Evolutionary computation is only effective when it maintains a good balance between *exploration* and *exploitation*. Exploration means searching widely in the space of candidate solutions (i.e. functions in our application). This is important for ensuring that we do not easily miss any good solutions. Obviously we shall not be able to explore every single individual, due to the combinatorial explosion problem explained in Section 3. Therefore, one has to invest one's computation time efficiently. This means spending more time on individuals that look more promising – if one can tell which individuals are more promising. In evolutionary computation, one hopes that the higher an individual's fitness is, the more promising that its genetic building blocks are in forming good offspring. Therefore, fitter individuals are given more chances to become parents. This is called exploitation (of the fitness information). Exploitation is implemented through tournament in EDDIE.

Exploration is closely related to diversity. It is important that some weaker individuals also have a chance to survive as diversity plays a key part in evolution (which is agreed by the experience in evolutionary computation). Diversity is achieved through the random operations in EDDIE; e.g. in picking individuals to participate in tournaments.

In EDDIE, functions are expressed in a grammar, which is specified by the user. This grammar must be defined in a formal, unambiguous way; typically in Backus-Naur Form in computer science. Figure 2 shows a grammar that a version of EDDIE has used; it has been varied in different versions of EDDIE.

Defining the grammar requires financial expertise. The engineer must know what indicators are relevant to target in the forecast. Garbage-in-garbage-out certainly applies here. If EDDIE is provided with indicators that are irrelevant to the price changes, then EDDIE will not be able to find rules that reflect opportunities.

```
                Tree  :=  "If-then-else"  <Condition>  <Tree>  <Tree> | <Prediction>
         <Condition>  :=  <Condition> "And" <Condition>  |  <Condition> "Or" <Condition>  |
                          "Not" <Condition>  |  <Indicator> <RelationOperation> <Threshold>
         <Indicator>  :=  technical indicators, such as moving average and break out rules, which
                          will not be elaborated here (see [27])
 <RelationOperation>  :=  "<"  |  ">"  |  "="
         <Threshold>  :=  Number
        <Prediction>  :=  "Positive" | "Negative"
```

Figure 2: The BNF grammar used by FGP

The grammar in Figure 2 only allows numerical relations such as "less than", "greater than" or equal to". There is nothing to stop one allowing more operators. For example, one can construct expressions from the given indicators using arithmetic operators (addition, multiplication, etc). Changing the grammar is not difficult at all. A more complex grammar will allow the program to express more complex functions, which in principle increases its chance of capturing good rules. In practice, a more expressive grammar demands the program to search a bigger space of functions. This tends to require more computation to achieve the same level of thoroughness in the search.

EDDIE finds interactions between indicators, and thresholds for those indicators. It is worth noting that interactions can also be found in other machine learning techniques such as neural networks [1]. Interactions found by neural networks are largely quantitative – they are basically numerical functions. Interactions found by genetic programming are qualitative – they are basically rules. Arguably rules are easier to understand by human users. This is important because it allows human experts to inspect the rules, and have a final say upon their meaningfulness. By being able to interpret the decisions, human expert can also channel their expertise into EDDIE, as Tsang et al [28] explained. By inspecting the rules and their performances, human experts can decide to modify relevant indicators, or add new indicators related to those which appear in successful rules.

9. Deeper inside the box – constraint-guided search

There is no magic in EDDIE, or machine learning in general. All an algorithm attempts to do is to search the space of solutions efficiently. The cleverer an algorithm, the more chance it has in finding good solutions. The efficiency of an algorithm can sometimes be improved by *heuristics*, which are guesses or rules of thumb that often work, though could be wrong – for example a restaurant that is full of customers tend to be better than one next door that has few customers, though there are numerous possible explanations beside food quality or value for money. In this section, we introduce one stipulation that gave EDDIE its efficiency.

Evolution is driven by the fitness measure. From research in constraint satisfaction (a field of computer science, e.g. see [25][20]), we know that constraints can help in a search, although it is the existence of constraints that created the problem in the first place. An experienced constraint programmer will use constraints to guide the search. Perhaps constraints can help EDDIE to focus its search in more promising areas. This was what Li and Tsang did in FGP [13] (which stands for Financial Genetic Programming), a version of EDDIE.

Based on past experience as well as the investor's expertise, one can often make a reasonable guess on how many investment opportunities there will be in a particular stock. In FGP, the user can prescribe the desirable range of opportunities, e.g. 10-15% of the time. This range forms a constraint to FGP. Decision trees that violate this constraint are punished. The hope is that by prescribing FGP a lower range, FGP will be more cautious in reporting opportunities. (Obviously if it is too cautious, it is never going to report any opportunities.)

Li and Tsang [13][27] found that the range set in the constraint did not affect FGP's overall correctness. However, it did change the precision and recall of the trees generated. When FGP was given a lower range (e.g. 5-10%) the trees generated had higher precision, but very low recall. When FGP was given a higher range (e.g. 30-50%) the trees generated had lower precision but higher recall. Therefore, the constraint gives the user a handle to trade precision with recall. This makes FGP more useful as a forecasting tool.

It is worth mentioning that, given the successful experience in FGP, Jin & Tsang [8][9] applied the constraint-guided incentive method to evolve strategies in automated bargaining [15]. The results were in line with theoretical results known. Evolutionary computation allows one to handle situations far more complex than those tackled theoretically so far [6].

10. How can we trust a program?

EDDIE is only a bit smarter than its human user. The human user is responsible for providing EDDIE with relevant financial indicators. If two investors are using EDDIE, the one who can provide EDDIE with smarter financial indicators are likely to out-perform the other.

What if the market is unpredictable? This will be the case if the market is efficient. Even if the market is inefficient, it can change fundamentally (paradigm change), which means past patterns are irrelevant to current behaviour of the market. Or, the grammar may not be sufficient to express the patterns in the market. In all these situations, EDDIE will report no patterns, as it often did. If the market is unpredictable, EDDIE will not make an investor poorer.

What if EDDIE gets it wrong, even if its human user provides it with all the relevant indicators? The fact is: it surely could. Nobody can be sure about the future. But that is not to say that all information that carries an element of uncertainty is useless. When paradigm changes occur, an investor who cannot recognize such changes will suffer the same with or without EDDIE. When patterns repeat themselves in the market, an investor could do better with support by EDDIE.

It is worth saying a word about automated trading. EDDIE can be embedded in auto-trading systems. Computer traders may not be cleverer, but are certainly more diligent than their human counterparts. While their human user needs to rest, eat and sleep, computer traders don't. EDDIE and similar programs can plough through financial data nonstop day and night. If one can afford more computers, one can run multiple copies of EDDIE. They can work on different targets. They will prompt their human users whenever opportunities are spotted. Computer traders will become more and more popular. Computer traders will not replace human traders, but they will work with them.

11. EDDIE-ARB predicts risk-free investment opportunities

EDDIE is not limited to evolving technical trading strategies. It can be supplied with fundamental values. In this section, we shall explain how EDDIE was used for detecting risk-free opportunities [29].

The prices of option and futures of a stock both reflect the market's expectation of futures changes of the stock's price. Their prices normally align with each other within a limited window. When they do not, *arbitrage* opportunities arise. This means an investor who spots the misalignment will be able to buy (sell) options on one hand, and sell (buy) futures on the other and make risk-free profits (this is known as *put-call-futures arbitrages*, details of which is explained in [29]).

According to the efficient market hypothesis, profitable risk-free investment opportunities should not exist. However, through analysing historical option and futures prices on the London International Financial Futures and Options (LIFFE) market, Hakan Er discovered that these prices occasionally do not align (around 15,500 cases from January 1991 to June 1998) [29]. Profitable risk-free investment opportunities do exist after transaction costs (assuming 0.1%) are taken into consideration. Judged by transaction record, a good proportion of these opportunities were indeed taken up. Arbitrage opportunities typically last for minutes or seconds only before the market adjusts itself.

Computer systems could be set up to monitor the market. As soon as price misalignment occurs, actions can be taken for profit. Such systems do not have to be equipped with any computational intelligence. All they need is life-feed of option and future prices from the market and a mathematical formula from economic textbooks that determines whether price misalignment has occurred. However, if two computer systems are set up to pick up the same arbitrage opportunities, then the competition is boiled down to which system has the faster network.

Could one detect such opportunities one step ahead of other arbitragers? This is where forecasting comes in. Prices do not become misaligned suddenly without early signs. EDDIE was fed with option and futures prices and the calculation of arbitrage profitability in the current market. For convenience, we call this specialization of EDDIE EDDIE-ARB. EDDIE-ARB was asked to forecast opportunities five minutes before they occur (to allow for delay in setting up the transactions).

Future and Options prices as they came did not help EDDIE-ARB to find patterns. They must be pre-processed to form variables suitable for building functions for forecasting. There is no magic in EDDIE, as we explained in Section 8. The more effective the pre-processing, the less hard work EDDIE-ARB has to do, and the more chance it has in finding quality rules.

As a tool, EDDIE-ARB enabled economists and computer scientists to work together to identify relevant function variables. Trained on historical data, EDDIE-ARB was capable of discovering rules with high precision. Tested on out-of sample data, EDDIE-ARB out-performed a nave ex ante rule, which reacts only when misalignments were detected [29]. This establishes EDDIE-ARB as a promising tool for arbitrage chances discovery. It also demonstrates how EDDIE brings domain experts and computer scientists together.

Although EDDIE-ARB has successfully forecast arbitrage opportunities, it only managed to pick up a very small proportion of the opportunities. The more cautious we instruct EDDIE-ARB to be (by using the constraint described in the Section 9), the fewer opportunities it picked up. Unfortunately, relaxing the constraint compromises precision. This limits the commercial potential of EDDIE-ARB, which motivates the research described in the next section.

12. When the going gets tough – facing scarce opportunities

Suppose I am asked "will there be an earthquake in the next three days?" It is pretty safe for me to answer "no". If I were asked the same question tomorrow, I could easily answer "no" again, regardless of my knowledge in earthquake. The chance of me getting it right is high, because earthquake does not happen too often, even if I live in an earthquake zone. However, although statistically my predictions are accurate, they are not interesting at all, because they fail to pick up any earthquake events.

In financial forecasting, one is keen to spot scarce events, such as arbitrage opportunities, big rises and crashes. Martinez-Jaramillo [14] pointed out that EDDIE could fail to find useful rules when it is given the task to find exceptionally high return within a short period of time. This does not just apply to EDDIE. It applies to practically all machine learning methods in the literature so far. They could fail because the data set is highly imbalanced. If very few of the instances are positive, there is not sufficient incentive for a function to classify any case as positive.

Consider the confusion matrix in Table 4. In reality, there are 9,900 (99%) negative instances, and 100 positive instances (1%). A conservative classification function that makes no positive classifications will be 99% correct; though its recall is 0% (precision is undefined as no positive classifications were made). As explained in the earthquake example above, this function is useless even though it is correct most of the time.

Table 4: Performance of a conservative classifier in an unbalanced data set
Accuracy: 99%; Precision: undefined; Recall: 0%

	False in Classification	True in Classification	Total
False in reality	True Negative: 9,900	False Positive: 0	9,900
True in reality	False Positive: 100	True Positive: 0	100
	10,000	0	

Suppose, knowing that 1% of the instances are positive, a classifier randomly make 1% positive classification. Its statistical performance is shown in Table 5. Compared with the conservative classifier, this classifier will have an improved recall rate (from 0% increased to 1%). However, this is paid for by deteriorating overall correctness (from 99% to 98.02%).

Table 5: Performance of a random classifier in an unbalanced data set

Accuracy: 98.02%; Precision: 1%; Recall: 1%

	False in Classification	True in Classification	Total
False in reality	True Negative: 9,801	False Positive: 99	9,900
True in reality	False Positive: 99	True Positive: 1	100
	9,900	100	

To do better than random classification, one would hope for a higher increase in True Positive than False Positive, such as the performance shown in Table 6. Like the random classifier, this classifier has made 1% positive classifications. However, it has better precision and recall (both at 10%, as opposed to 1% in the random classifier). Being able to pick up 10% of investment opportunities with exceptionally high return is not a bad thing. Unfortunately, this classifier has an overall accuracy of 98.2%, which lower than that of the conservative classifier (99%). Unless appropriate mechanisms are set up to encourage its survival, it will be discouraged or discarded in evolutionary computation. One could attempt to use a weighted fitness function, which gives higher weights to recall and lower weights to overall accuracy. Our experience with EDDIE suggests that this sometimes works, but sometimes encourages the wrong sort of classifiers. Performance of the classifier is very sensitive to the fine tuning of the weights.

Table 6: Performance of better than random classifier in an unbalanced data set

Accuracy: 98.2%; Precision: 10%; Recall: 10%

	False in Classification	True in Classification	Total
False in reality	True Negative: 9,810	False Positive: 90	9,900
True in reality	False Positive: 90	True Positive: 10	100
	9,900	100	

Motivated by the need to detect scarce opportunities, Alma García-Almanza invented the *Repository Method* [4][5]. It is designed to work with genetic programming, though it can be applied to any other methods that generate decision trees, such as C4.5 [18] or See5/C5.0 [19].

The Repository Method is built upon a set of carefully designed ideas. The basic principle is to analyze trees that are generated by genetic programming, with the view to (a) prune redundant or detrimental rules and (b) combine the strength of different rules. Each tree represents a number of rules, as illustrated in Section 7. There is no reason why one should take a tree in its entirety, when the performance of each of its rules can be examined against the data based on which the tree is generated. Furthermore, there is no reason why one should use only one tree for classification, when different rules in different trees could complement each other. The Repository Method maintains a repository of rules; starting from an empty set. A rule will be added to the repository if (a) it has high precision; and (b) it is dissimilar to the rules present in the repository. Similarity between two rules is measured by calculating the potential of them covering different instances in the database.

In evolutionary computation, the users often pick the best solution from the final generation. One common threat in evolutionary computation is that vital genetic material could be lost during the evolution process – a threat that many researchers are aware of and guard against. By using the Repository Method, one can combine rules from different generations. This alleviates (though does not eliminate) the problem of losing vital genetic material in the process. Besides, the rules generated during evolution will not go to waste. Experience shows that many of those trees during the early evolutionary process did contribute to the repository [4][5].

When opportunities are scarce, one does not have the luxury to ignore any useful rules. By combining useful trees from different generations, the Repository Method gives the user a better chance of picking up scarce opportunities. It is a useful tool for machine learning in highly unbalanced data set – in fact it is the only tool that the author is aware of that specializes in machine learning in highly unbalanced data sets. Experience shows that given a set of decision trees, the Repository Method will improve their performance reliably. It is able to pick more arbitrage opportunities than EDDIE-ARB (training on the same data set), without compromising precision.

13. Computational intelligence determines effective rationality

Rationality is a key assumption in all major classical economic theories, including the efficient market hypothesis mentioned in Section 2. Economic agents, whether it is a human being or a computer program, is assumed to maximize return and minimize risk. For simple decisions, such as choosing between \$100 and \$200, one can assume that a rational agent would choose the latter. However, when the situation is more complex, it is questionable whether the rationality assumption stands. For example, it is unreasonable to assume that an agent can always find the optimal move in a chess game.

Herbert Simon pointed out that most people are only partly rational. He suggested that people are "bounded rational", which means that they can only make the best decisions within their knowledge and resources [23]. Although most economists would accept that perfect rationality is not a realistic assumption, it is not clear how most of the economic theories can be revised to reflect bounded rationality. Concretely quantifying what bounded rationality means remains a grand challenge to the research community. As decision makers have to make decisions about how and when to decide, Ariel Rubinstein proposed to model bounded rationality by explicitly specifying decision making procedures [21]. This puts the study of decision procedures on the research agenda.

From a computational point of view, decision procedures can be encoded in algorithms and heuristics. In writing a chess program, some heuristics will find us better solutions than others. Our knowledge of algorithms, heuristics and our computational power determines how optimal our solutions can be. If rationality is measured by optimality, then our computational knowledge determines how rational we are. Therefore, designing better algorithms and heuristics helps to extend the rationality boundary. Computational Intelligence Determines one's Effective Rationality – we refer to it as the CIDER Theory [30].

Everything being equal, the quality of the solution depends on the algorithm that one uses. Some algorithms are better than others. From a computational point of view, we assume that an agent’s rationality is reflected by its computational intelligence.

An investor who uses EDDIE is likely to pick up more opportunities than the same user not using EDDIE. That is, EDDIE enhances this investor's rationality. The better forecasting algorithms one has, one would be able to pick up more opportunities. Therefore, advancing forecasting algorithms can be considered pushing back the boundary of rationality.

14. Where does it go from here?

Computational finance and economics is still in its infancy. Much more exciting research is lying ahead. When financial insight is combined with advanced computing, one can achieve what others cannot with financial knowledge or computing knowledge alone, as it is demonstrated in EDDIE-ARB.

History also shows that technology is often driven by application needs (e.g. warfare drives technology – this is a factual statement as opposed to a promotion of warfare). FGP and the Repository method both born out of needs that could not be fulfilled by current technology. The potential rewards for forecasting will motivate top-class computer scientists to invent more powerful computational methods.

It is important to point out that technology alone is not sufficient. A complete investment strategy demands more than a forecasting algorithm. For example, suppose we believe that prices of an asset will rise by 6% within the next 30 days. Do we commit all our capital on that asset? If we do, then when the program predicts another opportunity, we shall not have capital to invest in it. What proportion of one's capital should one invest when the program spots an opportunity? Financial expertise is needed to complement machine learning.

Forecasting algorithms so far are only scratching the surface of the matter. Most forecasting algorithms are trained on past share prices. The prices are in fact the result of interactions between traders. Traders interact through market orders, which are available for analysis. Research in this direction has barely started. If one studies the interactions, one could better understand the price dynamics, e.g. how big orders affect the prices (see [17]). Only when enough observation is available shall we be in a position to model decision processes or model market dynamics.

The field of computational finance has grown rapidly in the last few years. Judged by the number of publications, forecasting is a very popular research area in computational finance. Given the diversity of research (which is a healthy sign) it is difficult to know where the next breakthroughs will come; innovations are hard to foresee by nature. But it is easy to forecast that computational intelligence will completely change the frontier of forecasting research. It is also easy to forecast that companies that ignore new technology will be left behind.

Acknowledgements

This paper summarizes research results by my co-researchers and myself. All works that involves my co-researchers have been published. James Butler invented EDDIE, which is extended by Jin Li. I extended the idea of constraint-directed search to automated bargaining, which Nanlin Jin perfected. EDDIE-Arb was the result of many hours' collaborative work with and Hakan Er and Sheri Markose (both economists). The Repository Method was invented by Alma García-Almanza. Tim Gosling, Serafin

Martinez-Jaramillo, Biliana Alexandrova-Kabadjova and Paul Yung have all made valuable contributions to the research. The author is also grateful to the anonymous reviewers for their valuable advices.

References

[1] Bishop, C.M., Neural networks for pattern recognition, Oxford University Press, 1995

[2] Brock, W., Lakonishok, J. & LeBaron, B., Simple technical trading rules and the stochastic properties of stock returns, Journal of Finance, 47, 1992, 1731-1764.

[3] Fama, E.F., Efficient capital markets: A review of theory and empirical work, Journal of Finance, Vol.25, No.2, 1970, 383-417

[4] García-Almanza, A., New classification methods for gathering patterns in the context of genetic programming, PhD Thesis, Department of Computing and Electronic Systems, University of Essex, July 2008

[5] García-Almanza, A.L. & Tsang, E.P.K., Detection of stock price movements using chance discovery and genetic programming, International Journal of Knowledge-based and Intelligent Engineering Systems, Vol.11, No.5, December 2007, 329-344

[6] Gosling, T., Jin, N. & Tsang, E.P.K., Games, supply chains and automatic strategy discovery using evolutionary computation, in J-P. Rennard (Eds.), Handbook of research on nature-inspired computing for economics and management, Vol II, Chapter XXXVIII, Idea Group Reference, 2007, 572-588

[7] Holland, J.H., Adaptation in natural and artificial systems, University of Michigan press, Ann Arbor, MI, 1975

[8] Jin, N., Equilibrium selection by co-evolution for bargaining problems under incomplete information about time preferences, Proceedings, Congress on Evolutionary Computation, Edinburgh, 2-5 September 2005, 2661-2668

[9] Jin, N. & Tsang, E.P.K., Co-adaptive Strategies for Sequential Bargaining Problems with Discount Factors and Outside Options, Proceedings, Congress on Evolutionary Computation (CEC) 2006, 7913-7920

[10] Koza, J.R., Genetic Programming: on the programming of computers by means of natural selection, MIT Press, Cambridge, MA, 1992

[11] Koza, J.R., Genetic programming II: automatic discovery of reusable programs, MIT Press, Cambridge, MA, 1994

[12] Li, J. & Tsang, E.P.K, Investment decision making using FGP: a case study, Proceedings of Congress on Evolutionary Computation (CEC'99), Washington DC, USA, July 6-9 1999.

[13] Li, J. & Tsang, E.P.K, Reducing failure in investment recommendations using genetic programming, Computing in Economics and Finance Conference, Barcelona, July 2000

[14] Martinez-Jaramillo, S., Artificial financial markets: an agent based approach to reproduce stylized facts and to study the Red Queen Effect, PhD Thesis, Centre for Computational Finance and Economic Agents (CCFEA), University of Essex, 2007

[15] Muthoo, A., Bargaining theory with applications, Cambridge University Press, 1999

[16] Neely, C., Weller, P. & Ditmar, R., Is technical analysis in the foreign exchange market profitable? A genetic programming approach, Journal of Financial and Quantitative Analysis, 32, 1997, 405-26

[17] Olsen, R., Classical economics: an emperor with no clothes, Wilmott Magazine Volume 15, January 2005, p84-85

[18] Quinlan, J.R., Improved use of continuous attributes in C4.5, Journal of Artificial Intelligence Research, AI Access Foundation and Morgan Kaufmann Publishers, Vol.4, 1996, 77-90

[19] Quinlan, J.R, Data mining tools See5 and C5.0, http://www.rulequest.com/see5-info.html (accessed 25 August 2008)

[20] Rossi, F., van Beek, P. & Walsh, T. (ed.), Handbook of Constraint Programming, Elsevier, 2006

[21] Rubinstein, A., Modeling bounded rationality, MIT Press 1998

[22] Shleifer, A., Inefficient markets, an introduction to behavioral finance, Oxford University Press, 2000

[23] Simon, H., Models of Man, Wiley, New York, 1957

[24] Sweeney, R.J., (1988), Some new filter rule tests: Methods and results, Journal of Financial and Quantitative Analysis, 23, 285-300.

[25] Tsang, E.P.K., Foundations of constraint satisfaction, Academic Press, London and San Diego, 1993

[26] Tsang, E.P.K., Butler, J.M. & Li, J., EDDIE beats the bookies, International Journal of Software, Practice & Experience, Wiley, Vol.28(10), 1998, 1033-1043.

[27] Tsang E.P.K. & Li, J., EDDIE for financial forecasting, in S-H. Chen (ed.), Genetic Algorithms and Programming in Computational Finance, Kluwer Series in Computational Finance, 2002, Chapter 7, 161-174

[28] Tsang, E.P.K., Yung, P. & Li, J., EDDIE-Automation, a decision support tool for financial forecasting, Journal of Decision Support Systems, Special Issue on Data Mining for Financial Decision Making, Vol.37, 2004, 559-565

[29] Tsang, E.P.K., Markose, S. & Er, H., Chance discovery in stock index option and future arbitrage, New Mathematics and Natural Computation, World Scientific, Vo.1, No.3, 2005, 435-447

[30] Tsang, E.P.K., Computational intelligence determines effective rationality, International Journal on Automation and Control, Vol.5, No.1, January 2008, 63-66