3

MACHINE LEARNING IN FINANCE

3.1 MACHINE LEARNING FOR FORECASTING

So far, we have introduced the synergy between computing and finance in algorithmic trading. We have also explained the success and limitations of machine learning. In this section, we shall take a closer look at an application of machine learning in finance: forecasting.

Forecasting is an important subject in finance. The hope is to predict what is going to happen based on what has been observed so far. We shall focus on the following forecasting target, which we introduced in Chapter 1:

> *Forecast Target 1: "Will the FTSE 100 Index rise by 4% within the next 7 days?"*

The first step in applying machine learning to forecasting is to define the target. In the above example, the target is to predict whether "the price will rise by 4% within the next 7 days". In technical terms, the target of this forecast is to predict the value of a Boolean variable, which could take the value "true" or "false".

Readers are reminded that the forecasting target does not have to be a Boolean variable. One could try to forecast:

> *Forecast Target 2: "What will the closing FTSE 100 Index be tomorrow?"*

DOI: 10.1201/9781003348474-3

In this case, the variable to be forecasted is a number.

After deciding what to forecast, the second step is to identify the variables that may help us to forecast the target. For example, technical analysts believe that future prices can be predicted from the past. In order to predict the value of the FTSE 100 Index tomorrow, they may feed a machine learning system with, say, the past 10 years' Index values. Fundamental analysts will use fundamental information, such as interest rates, exchange rates, consumer price index, trade surplus or deficit, etc., for forecasting.

Another decision to make is to decide on what machine learning method to use to predict the target with the input variables. Artificial neural networks were used in AlphaGo. Both artificial neural networks and genetic programming have been used in financial forecasting. Apart from these two, many other machine learning methods have been invented. Different techniques are suitable for different applications, so knowledge in which method works for what problems is important. This is a non-trivial topic and ongoing research, which is way beyond the scope of this book.

In the next section, we shall explain how machines might learn to predict the target values. Computer scientists call the form of learning that we are about to introduce "supervised learning". This is because it requires a training session, in which the trainer must tell the system what the correct target value should be for each set of input data. Later in this chapter, we shall introduce "unsupervised learning", in which no training is required.

3.2 SUPERVISED LEARNING

A few technical terms should help understand what machine learning is about: The input variables are called "independent variables". The target is called a "dependent variable", as for machine learning to work, the target's value must be dependent on the value of the input variables. If it is not, then machine learning will never find anything useful. Machine learning's task is to find the dependency

relationship. That means finding out what the target's value should be given a set of input values.

In supervised learning, the trainer tells the machine learning system what the correct target value should be under a given set of input values. How would the trainer know what the correct output should be? That is normally done through hindsight. By looking back 7 days, one knows "whether the price will rise by 4% within the next 7 days" for "Forecast Target 1" above. For example, by looking back 1,006 days, one gets 1,000 sets of correct input–output relationships.

When the trainer provides the machine learning system with an input and tells it what the correct output should be, the system will adjust its internal parameters to guide it towards giving the correct answer in the future. That is how learning progress is made. Normally, there are many ways to adjust the internal parameters towards the given answer. There is no guarantee that the system will make the right adjustments. That is why the system will guide itself towards the correct answer rather than make sure that it gives the correct answer. This gradual learning strategy is used by most learning systems. It is also needed to allow the data to be noisy (sometimes the "correct answer" may not be correct in other cases; it could be an exception), which financial data often are.

The training is repeated with the same data. Every time the training data is passed through the machine learning system, it adjusted its parameters a bit. When training is completed, the output is expected to match most of the correct answers. This should be the case if the value of the target is indeed dependent on the input values.

Mathematically minded readers may see supervised learning as a "function-fitting" exercise. Learning is conducted through calibration. The machine learning system attempts to find a function that maps the input to the output. This function could take any form. In an artificial neural network, the function takes the form of a mathematical relationship between the input and the output via some intermediate internal variables. The neural networks that AlphaGo uses take many layers of internal variables, as shown in Figure 3.1,

hence the term "deep learning". The input nodes (on the left of Figure 3.1) pass their values to the nodes in the first hidden layer next to it via the connections. Each connection carries a "weight", which adjusts the strength of one node to another. These weights are adjusted through training. The value of an internal node is the weighted sum of all its inputs. So, the whole network is a mathematical function from the input to the output.

With "genetic programming", another paradigm of machine learning, logical relations could be handled more easily. That means it can learn rules in the form of: "if the 7-day Moving Average was below the 21-day Moving Average yesterday, but the relationship is reversed today, then the price will rise tomorrow, otherwise …".[1] So, genetic programming can handle logical functions from the input to the output.

Anyone who uses supervised learning for financial forecasting must remember that they are making an important assumption: the behaviour of the market in the future is similar to the market's behaviour in the past. This is because training is conducted with historical data. If market behaviour changes, there is no guarantee that the forecasting function learned will apply to the market in the future.

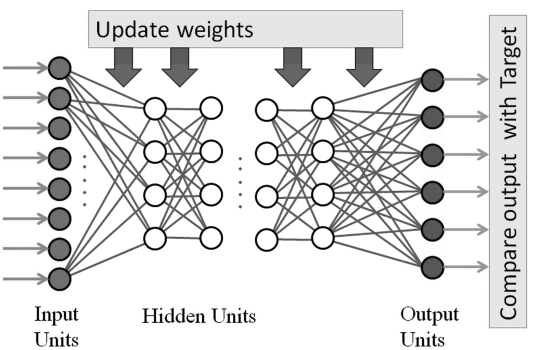

Figure 3.1 Structure of a Multi-layer Artificial Neural Network.

When would the market change its behaviour? New financial instruments play an important part: when futures and options were introduced, they gave traders more tools to conduct their trades. Technology plays an important part too. Algorithmic trading increases trading frequency dramatically. Political events change the traders' behaviour too. For example, when Brexit results were announced, the foreign exchange market temporarily behaved differently from before. These were the moments when the machine-learned forecasting may not have worked.

3.3 KNOW YOUR DATA

Machine learning algorithms are important, but supervised learning will not work unless it has data – not just in quantity, but in quality.

Big data matters: in general, the more data machine learning uses, the better the result tends to be. Supervised learning is basically a generalization exercise. It searches for patterns that are supported by data, with the hope that these patterns repeat themselves in the future. Generalizing from ten examples is dangerous. Generalizing from 10,000 examples is better. Everything being equal, the more data one uses, the more reliable the learned patterns.

However, "everything being equal" must be examined carefully. Suppose one uses daily closing prices for forecasting (discussed in Section 3.1). Using historical data for 750 days (about three years) for training is probably better than using 250 days. But is it a good idea to use 10,000 days (about 40 years)? Probably not. This is because the market from the financial crisis in 2007 and 2008 was very different from the market in more recent years. Patterns learned from that period may never repeat themselves in the future. Therefore, the inclusion of 2007–2008 data may not benefit machine learning.

Besides, the market has changed significantly. As mentioned above, algorithmic trading has grown in popularity; they trade differently from human traders. Besides, more financial instruments have been introduced. Regulations have changed, which affects the behaviour of corporate investors. All these make old

data less relevant to today's market. Therefore, more data is not always better.

While we are on this topic, it is probably worth explaining why learning from "higher-frequency data" (such as minutely prices) is better than using "lower-frequency data" (such as daily closing prices). If one uses daily closing data, there are approximately 250 data points per year. If one uses minutely closing data, one has access to approximately 120,000 data points per year, nearly 500 times more. High-frequency finance will be elaborated in Section 7.2.

While having sufficient data is essential, the quality of data is critical to machine learning. In Section 1.5, we mentioned the lipstick index. Is it possible to use lipstick sales to forecast GDP growth in the next quarter? Correlations between lipstick sales and GDP growth may have been found in the past, but they do not hold all the time. To be able to forecast reliably, the variables used for prediction must be relevant to the target – the higher the relevance, the better.

Machine learning is no magic. How one prepares the data affects machine learning's ability to find patterns. For argument's sake, suppose the 7-day moving average is an important indicator which helps to forecast. If we input to the program the past 1,000 days' closing prices, machine learning could learn to use the 7-day moving average by itself. But if we pre-compute the 7-day moving average and supply it as an input variable to the program, then we increase the machine's ability to learn useful patterns using this variable.

To summarize, the data that one uses affects a machine learning program's effectiveness and efficiency. What a program may potentially learn depends on how financial data are collected and how we present the data to the program. We shall revisit the financial data issue in Chapter 6.

An old saying in computer science is always worth remembering:

"Garbage in, Garbage out!"

Readers should note that this point in no way contradicts with AlphaGo Zero's idea of general intelligence (Section 2.2). The

creator of AlphaGo Zero fed no intelligence about the game into the program. But AlphaGo Zero received all the input that is relevant to the game (the current state of each position). So AlphaGo Zero did not start with garbage; it started with all the necessary input for learning.

3.4 A GLIMPSE OF GAME THEORY

In this and the next section, we shall turn our attention to unsupervised learning. We shall use bargaining theory as an example to demonstrate how unsupervised learning works.

In supervised learning, the trainer must tell the program what the "correct" solution should be. For example, in the board game Go, AlphaGo used supervised learning to start. It used games played by top human players to show the program where good moves are. This allowed AlphaGo to conduct supervised learning at its initial stage. Following the success of AlphaGo, AlphaGo Zero was developed. There, supervised learning was dropped. This makes sense, as AlphaGo was already beating top human players. Even if supervised learning were to be conducted, AlphaGo Zero should have been shown AlphaGo's moves, not human players' moves.

Unsupervised learning is conducted when we cannot tell (or, in AlphaGo Zero's case, do not want to tell the program) what the target solution is. It does not matter whether we know what a good solution is, as long as we know whether a solution is good or bad when we see it. In the game of Go, we know a good program is one that wins more games than it loses. This is sufficient for unsupervised learning to apply. In this section, we shall introduce a bargaining problem. In the next section, we shall explain how unsupervised learning can be applied to the bargaining problem.

Bargaining is one of the two most studied areas in game theory (the other being repeated games, such as the Prisoner's Dilemma). In the next section, we shall look at how unsupervised learning could be applied to bargaining. Before that, we shall introduce a bargaining model.

Following is a textbook scenario in bargaining:

> **Basic Alternating-Offers Bargaining Model**
> Players A and B are about to share a pie. A makes an initial offer, offering to share a certain percentage with B. B may reject the offer, in which case, B will make a counteroffer to A. Then it is A's turn to decide whether to accept or reject the offer. The bargain continues until one side accepts the offer or no deal is struck. To give both parties an incentive to make reasonable offers and accept offers as soon as possible, the game stipulates that the utility of their shares drops increasingly over time. Importantly, both parties know how fast both utilities drop. The player whose utility drops more slowly would have an advantage in the bargaining.

Here is a bargaining scenario:

> Turn 0: A offers 20% of the pie to B.
> If B accepts the offer, then A will have a utility of 80% and B 20%.
> Turn 1: B rejects A's initial offer; B counteroffers 50% to A.
> If A accepts the offer, then A will have a utility of 27% (not 50%, as the utility drops) and B will have a utility of 34% (assuming that B's utility drops more slowly)
> Turn 2: A rejects B's 50% offer; A counteroffers 40% to B.
> If B accepts the offer, then B will have a utility of 18% (discounted from the 40% being offered) and A 18% (discounted from the 60% that A retains)
> …

A little reflection should convince the readers that A would have been irrational to reject B's offer of 50% in Turn 1 if A planned to counteroffer 40% to B in Turn 2. This is because accepting the 50% in Turn 1 gives A a utility of 27%, which is better than getting 18%

(discounted from 60%) in Turn 2. A would have been better off taking the 50% offer from B in Turn 1.

To push the logic further, A should have made B a better initial offer (in Turn 0) had A anticipated that B will reject 20%. In order to mathematically work out what A's initial offer should be, bargaining theorists assume the following:

Assumption 1: Both players are fully rational, which means they can both make decisions that maximize their share of the pie (see discussion in Section 2.5). Both players know that their opponent is fully rational.

Assumption 2: Both players know the rates at which the two players' utilities drop over time (their utilities may drop at different rates). They also know what their opponent knows.

Under the above assumptions, bargaining theorists can work out A's initial offer that B cannot refuse. Both players will be better off agreeing upon the initial offer. Any delay in the agreement will discount their utilities. In other words, while the two players are competing for limited resources, they must also cooperate in order to maximize their rewards.

It is worth reminding computer scientist readers that the search for the initial offer is not a simple optimization problem. A computer scientist would be tempted to try one value at a time, from 0% to 100%, for a given precision in an attempt to find the optimal solution. However, it is not a simple problem of evaluating every value because how good an offer depends on the opponent's response, which in turn depends on the first player's subsequent response. Computationally, this is not dissimilar to a game of Chess or Go. Game theorists call the solution a "subgame equilibrium" (as opposed to an "optimal solution", which computer scientists are familiar with). To make the initial offer at Turn 0, Player A would ask itself what B would offer in Turn 1 if B rejects A's initial offer. In other words, Player A attempts to solve B's subproblem at Turn 1.

With that subgame solution, A would know what to offer in Turn 0. But then how would B solve the subproblem in Turn 1? A would anticipate that B would solve the subgame problem at Turn 2 from A's point of view. So the subgame equilibrium is solved recursively. When the reasoning is repeated recursively towards infinity, which is possible mathematically, the initial offer at Turn 0 can be solved.

3.5 "UNSUPERVISED LEARNING" FOR BARGAINING

In this section, we shall explain how unsupervised learning can be applied to the Basic Alternating-Offers Bargaining model introduced in the previous section. The approach that we are going to explain is based on "evolutionary computation", an idea that is borrowed from natural evolution. Given a problem, instead of designing and building solutions, one attempts to evolve solutions.

To apply evolutionary computation, candidate solutions must be represented using building blocks – think of them as genes. (Knowledge representation is an important part of AI. We shall revisit this issue in Chapter 6.) For this bargaining model, a candidate solution is a function made up of the two players' discount rates which determines how fast the two players' utility drops over rounds. Computer scientists are familiar with representing functions with trees. So a candidate solution can be represented by a tree which branches are made up of arithmetic operations ($+$, $-$, $\times$, $\div$), numbers and the given discount rates. Different combinations of these operations form different functions. A tree is a function that can be reduced to a number that represents a player's offer to their opponent. The task of machine learning here is to explore different ways to combine the building blocks.

The general principle of evolution is to maintain a population of candidate solutions and let them evolve good solutions: To start, a population of candidate solutions is generated randomly. The fitness of the individuals in the population is determined by how well they meet the requirement (in the case of optimization) or

solve the problem (in the case of problem-solving). The fitter an individual, the more chance it is given to pass its building blocks (genes) to future generations. The hope is that building blocks that contribute to fit individuals will be allowed to construct better solutions. Evolution ends when individuals in the population converge on similar solutions (this will be the case when the majority of the individuals use the same building blocks), or time runs out.

To apply evolutionary computation to the above bargaining model, a two-population approach can be used. Based on the solution representation described above, a population of strategies is generated for Player 1 and another population for Player 2, as shown in Figure 3.2. These two populations co-evolve through competition between the individuals. Individuals in the population for Player 1 will bargain with individuals in the population for Player 2. Successful individuals, namely, those that score high utilities will be encouraged to pass their building blocks to future generations.

This approach to learning bargaining strategies is a form of unsupervised learning. Unlike supervised learning, there is no trainer to tell the program what the correct solution should be. The candidate solutions find their fitness through playing against opponents. The designer's task is to design the representation of candidate solutions and the way to maintain evolutionary pressure to enable fit individuals to pass their building blocks (genes) to future generations.

It is worth iterating the point that if the programs are allowed to evolve solutions freely, many of the candidate solutions generated could be very poor bargainers. They could ask for over 100% of the

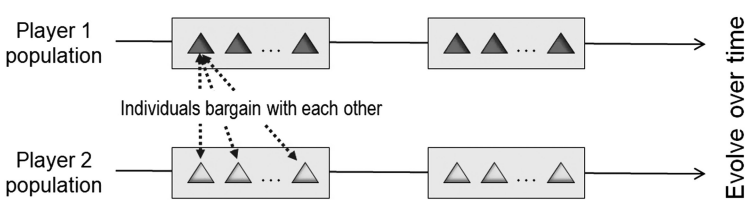

Figure 3.2 Co-evolution in Bargaining.

pie. They could also ask for a negative percentage of the pie. To help the search to focus, incentives or constraints can be added to the fitness evaluation step to help the search to focus on more promising solutions. Details of this approach are beyond the scope of this book.

There are at least three reasons why machine learning is an attractive approach to handling bargaining problems. Firstly, classical bargaining theory is a mathematical approach. It relies on the perfect rationality assumption. Human beings are not perfectly rational, as Nobel laureate Herbert Simon pointed out (discussed in Section 2.5). Human bargainers rarely can reason recursively towards infinity (as explained in the previous section). Instead of assuming perfect rationality, machine learning assumes reinforcement learning. This is arguably closer to human reasoning.

Secondly, a mathematical approach can only handle neatly defined mathematical problems. Unlike the Basic Alternating-Offers Bargaining model, real-life bargaining problem often involves messy relations, including logical and procedural operations, which makes mathematical analysis very difficult. A machine learning approach will handle logical and procedural operations all the same.

Thirdly, a slight alteration of the bargaining model could demand a completely new mathematical analysis. With evolutionary computation, one only needs to change the bargaining strategy representation. The evolutionary process is the same. For example, if Player B is ignorant about the utility deterioration rate of Player A, all one needs to do is remove A's utility deterioration rate from the language that defines Player B's strategy representation. With B's ignorance, a mathematical analysis would find the subgame equilibrium difficult to solve in the resulting model. With reinforcement learning, the evolutionary computation would be able to generate subgame equilibrium.

The approach that unsupervised learning uses is "generate-and-test". It generates candidate solutions and tests their fitness. New candidate solutions are generated based on the successful solutions found so far. Randomness almost always plays a part in the generation of new solutions. Randomness is important because the approach is

basically sampling in the space of solutions. "Generate-and-test" is an old AI term. It was forgotten because it does not sound as exciting and imaginative as other terms such as artificial neural network, evolutionary computation and other search methods that use names which suggest nature inspiration. But it describes what many search methods, including unsupervised learning, basically do.

3.6 SUMMARY: MACHINE LEARNING IS A GAME CHANGER

In the previous chapter, we explained the promise and limitations of machine learning. In this chapter, we have looked into two forms of machine learning: "supervised learning" and "unsupervised learning".

Supervised learning requires the trainer to tell the program what the correct target values are. For example, in forecasting, the trainer must tell the program what the correct forecast is. What supervised learning does is essentially function-fitting: the machine learning system uses the training material to calibrate a function that would produce a forecast from the input variables. For supervised learning to be successful, it is crucial to choose the right variables: the value of the target must be dependent on the value of the input variables.

Unsupervised learning does not require a trainer. Solutions are evolved rather than designed. It has been used by AlphaGo to play the game of Go; it has also been used to find subgame equilibrium in bargaining, a branch of game theory. What unsupervised learning does is essentially generate-and-test: successful candidate solutions are encouraged to pass their building blocks to future generations, with the hope that better solutions will be evolved over generations. For unsupervised learning to be successful, it is important to build a proper (artificial) environment for the individuals to interact within and a reliable assessment of an individual's fitness.

Machine learning is powerful. It can be a game changer. However, one must understand that there is no magic in machine learning. Before the General AI approach (described in Section 2.2) succeeds,

expertise is needed to help machine learning to succeed. Expertise is needed in determining what to learn, choosing the variables, designing candidate solutions, choosing machine learning methods or developing new ones if necessary.

NOTE

1. This is an example showing the form of a logical relationship between the input (which are 7-day moving average and 21-day moving average values) and the output (which is "will the price rise tomorrow?"). Readers are reminded that this is just an example, not a realistic rule.