Computational Intelligence in Economics and Finance: Shifting the Research Frontier

Shu-Heng Chen[1], Paul P. Wang[2], and Tzu-Wen Kuo[3]

[1] AI-ECON Research Center, Department of Economics, National Chengchi University, Taipei, Taiwan 11623, chchen@nccu.edu.tw
[2] Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA, ppw@ee.duke.edu
[3] Department of Finance and Banking, Aletheia University, Tamsui, Taipei, Taiwan 25103, kuo@aiecon.org

Summary. This chapter provides an overview of the book.

1 About the CIEF Series

This volume is the continuation of the volume with the same title which was published by Springer in 2003 ([18]), and is part of the same series of post-conference publications of the **International Workshop on Computational Intelligence in Economics and Finance** (**CIEF**, hereafter). The previous volume is mainly a collection of selected papers presented in CIEF 2002 (the 2nd CIEF), whereas this one is a collection of selected papers in CIEF 2005 (the 4th CIEF).

The idea of the CIEF was first initiated by Paul P. Wang, one of the editors of this volume. Reference [11] (p. 123) details the historical origin of the CIEF. Intellectually, the CIEF carries on the legacy of Herbert Simon, who broke down the conventional distinctions among economics, computer science and cognitive psychology, and initiated the interdisciplinary research field that we refer to as artificial-intelligence economics. The later development of CIEF, including not only its depth of coverage but also its breadth of coverage, are documented in [12].

The fourth CIEF was held as a part of the 8th Joint Conference on Information Sciences (JCIS 2005) between July 21-26, 2005 in Salt Lake City, Utah. Among the 15 tracks of JCIS 2005, CIEF 2005 is by far the largest one. A total of 18 sessions with 81 presentations were organized. Authors of the 81 papers were encouraged to submit their extended versions of the conference papers to the post conference publications. Twenty-seven submissions were received, and each of them was sent to at least two referees. In the end, only nine out of 27 submissions plus three additional invited papers were accepted. These constitute the contents of this volume. [1]

[1] Seven other papers were published in a special issue of *Journal of New Mathematics and Natural Computation*, Vol. 2, No. 3.

2 About this Volume

2.1 Structure of the Volume

To closely connect this volume with the previous one, we structure the book in a similar fashion compared with that of the previous one, i.e., the chapters are grouped and ordered by means of the computational intelligence tools involved. As the organization chart indicated in Fig. 1.1 ([18], p.4), the structure of the book is presented in the order of fuzzy logic, neural networks (including self-organized maps and support vector machines) and evolutionary computation. The same order is applied here. In other words, this volume is also structured by using the same organization chart in Vol. 1.

However, as a continued volume, this volume does not contain a comprehensive overview of the entire economic and financial applications of CI as was the case in the first volume. A number of techniques which can be seen in Vol. 1 are not presented here, including rough sets, wavelets, swarm intelligence (ant algorithms), and agent-based modeling. Nonetheless, there are also "new faces" appearing in this volume, including recursive neural networks, self-associative neural networks, K-means and instance-based learning.

Given the large degree of similarity to Vol. 1, there is no need for a voluminous introductory chapter as we saw in Vol. 1 ([18], Chap. 1). However, for the additional techniques which do not appear in the first volume, a brief introduction is provided. The brief introduction is not meant to be a tutorial, but mainly to show how this specific tool is related to some other tools, which we see in Vol. 1. For example, it shows how the recurrent neural network and the self-associative neural network are related to the feedforward neural network, how K-means is related to self-organizing maps and K-nearest neighbors, and how instance-based learning is related to K-nearest neighbors. In this way, we make an effort to make everything as tight as possible and leave the audience with a comprehensive understanding of the materials.

2.2 Themes

As the second volume, this volume shares a great many similarities, not only in techniques but also in terms of themes, with the previous volume.

Efficient Markets Hypothesis

A large part of this volume can be read as a continuous effort to question and to challenge the efficient markets hypothesis or the random walk hypothesis. Are stock prices predictable? Are trading strategies profitable? While these issues are old, they never die and they are still the central themes of these chapters. Even though financial econometricians nowadays also frequently address these issues, what distinguishes financial econometricians from CI researchers is the way in which they prove the answer.

Financial econometricians test the hypotheses based on *data with probabilistic models*, whereas CI researchers test the hypotheses based on *data with algorithms*. By assuming a probabilistic universe, econometricians like to question the data-generating mechanism and ask whether the data observed are indeed randomly generated. For CI researchers, we treat computational intelligence as an unbounded set of algorithms. These algorithms are intelligent in the sense that each of them articulates well *what the patterns are*, and, by applying these algorithms to data, we inquire of the existence of such kinds of patterns. New algorithms propose new ways of thinking about patterns and of constructing patterns, and hence new tests for the hypotheses. Chapters 3, 7, 8, 9, 11 and 12 are studies of this kind.

Nonlinearity

An issue related to the efficient markets hypothesis is nonlinearity. In fact, one fundamental question regarding the efficient markets hypothesis is whether financial time series are linear or nonlinear. Nonlinearity motivates the use of CI tools in many chapters of this volume. It, in effect, provides the connection between conventional statistics and CI tools. This point has been well illustrated by many chapters. Examples include the artificial neural network, particularly the recurrent neural network, as an extension of linear time series modeling (Chap. 3), the self-associative neural network as an extension of linear principal components analysis (Chap. 4), and the support vector machine as an extension of the linear classification model (Chap. 5). In each of these cases, the central issue is whether one is able to capture the neglected nonlinearity of the linear models from the proposed nonlinear counterparts.

The artificial neural networks and genetic programming are non-parametric and hence are quite flexible to different functional forms. This flexibility can be quite crucial because, as pointed out by [7], "...unlike the theory that is available in many natural sciences, economic theory is not specific about the nonlinear functional forms. Thus economists rarely have theoretic reasons for expecting to find one form of nonlinearity rather than another." (*Ibid*, p.475.)

Statistics and Computational Intelligence

Despite the great flexibility, a repeatedly asked question concerns the superiority of the non-linear models as opposed to linear models in forecasting (Chap. 3) or classification (Chap. 5). The performance of CI tools is therefore frequently compared with conventional statistical models. Chapters 3 and 5 provide two good illustrations.

The remainder of this introductory chapter provides a quick grasp of the 12 chapters included in this volume. As mentioned above, the 12 chapters will be briefly introduced in an order beginning with fuzzy logic (Sect. 3), artificial neural networks (Sect. 4), and then evolutionary computation (Sect. 5).

3 Fuzzy Logic

Fuzzy logic interests us because it enhances our flexibility in modeling the human's inner world. Quite often, we attempt to model human behavior by "fitting" their actual decisions to the observed external environment which is frequently characterized by a large number of features. In this way, a formal decision rule can be explicitly generated.[2] However, humans are not always confident about what they are doing or choosing. They often make decisions based on fears or skepticism. Without knowing this limitation, those decisions rules extracted from the observed data can be misleading.

While neural scientists, over the last decade, have been trying hard to make us know more about this process within the "box", it still largely remains "black" to us. Therefore, a sensible model of decision rules must, to some extent, respond to this "softness." Fuzzy numbers, fuzzy coefficients (in the fuzzy regression models) or fuzzy decision rules may be read as a way of coping with this reality, and there may be a neural foundation for fuzzy logic, which is yet to be established. Alternatively, we may ask: *is the brain fuzzy,* and *in what sense*? Or, we may say: fuzziness is everywhere, because it is in our brain.

For example, in the 1970s, there were some discussions between psycho-linguists and fuzzy theorists on the use of the adverb *very*. For an illustration, two different interpretations of *very large* arise. In one case, the fuzzy set *very large* is included in the fuzzy set *large*. In the other case, it is not; *large* and *very large* denote two different categories.[3] Perhaps by using the scanning technology of neural sciences, we can explore the relationship between *very large* and *large*, and hence provide a neural-scientific foundation for the membership function and its associated mathematical operation.

In Chap. 2, **An Overview of Insurance Use of Fuzzy Logic**, Arnold F. Shapiro provides a comprehensive review of the use of fuzzy logic in insurance, or the field known as *fuzzy insurance*. The unique writing style of the author makes this chapter suitable for readers with various levels of intellectual curiosity.

First, it obviously serves the readers who want to know the relevance of fuzzy logic to insurance. This chapter presents a great number of examples, ranging from risk classification, underwriting, projected liabilities, rate making and pricing, asset allocation and investments. Of course, due to the large number of examples, the authors are unable to give enough discussion to each single case. Therefore, for general audiences, they may experience a little difficulty quickly catching the hard lesson to be learned from each example. Nevertheless, a small taste of each dish in such a rich buffet does, at least, help us get the message that fuzzy logic is indispensable for insurance enterprises.

Secondly, in addition to application *per se*, the author has carefully further categorized the applications into groups from the perspectives of fuzzy logic. Starting

[2] This is how software agents are connected to human agents. For more details, see [12].

[3] When a listener hears that *x is large*, he assumes that *x is not* very large, because in the latter case the speaker would have used the more informative utterance *x is very large*. See [29] and [22].

from fuzzy sets and fuzzy numbers, the author concisely goes through fuzzy arithmetic, fuzzy inference systems, fuzzy clustering, fuzzy programming, and fuzzy regression. This, therefore, enables us to ask a more fundamental question: *Why fuzzy*? *What is the nature and the significance of fuzzy logic?* Certainly, we are not the first to ask and address the question, but the question is so deep and so important that it is worth our asking it and addressing the issue over and over again. This chapter places us in a specific daily life situation, i.e., a risky life, enabling us to revisit the issue. It drives us to think of the value of fuzzy logic while acting as if we are making insurance decisions in dealing with various risky real-life events, such as earthquake damage, health forecasts, etc.

4 Artificial Neural Networks

Among all the economic and financial applications, the artificial neural network (ANN) is probably the most frequently used tool. It has been shown in a great number of studies that artificial neural networks, as representative of a more general class of non-linear models, can outperform many linear models and can sometimes also outperform some other non-linear models.[4]

4.1 Recurrent Neural Networks

Three classes of artificial neural networks have been most frequently used in economics and finance. These are *multilayer perceptron neural networks*, *radial basis neural networks*, and *recurrent neural networks*. The first two classes were introduced in Vol. 1,[5] whereas the last one is introduced in this volume.

In Vol. 1, we discussed the relationship between time series models and artificial neural networks. Information transmission in the usual multilayer perceptron neural network is *feedforward* in the sense that information is transmitted *forward* from the input layer to the output layer, via all hidden layers in between, as shown in Fig. 1. The reverse direction between any two layers is not allowed.

This specific architecture makes the multilayer perceptron neural network unable to deal with the moving-average series, $MA(q)$, effectively. To see this, consider an $MA(1)$ series as follows.

$$x_t = \epsilon_t - \theta_1 \epsilon_{t-1}. \tag{1}$$

It is well-known that if $\mid \theta_1 \mid < 1$, then the above $MA(1)$ series can also be written as an $AR(\infty)$ series.

$$x_t = -\sum_{i=1}^{\infty} \theta^i x_{t-i} + \epsilon_t \tag{2}$$

[4] This is not a good place to provide a long list, but interested readers can find some examples from [4], [24], [28], [43], [44], [45], and [47].

[5] See [18], pp. 14–18.

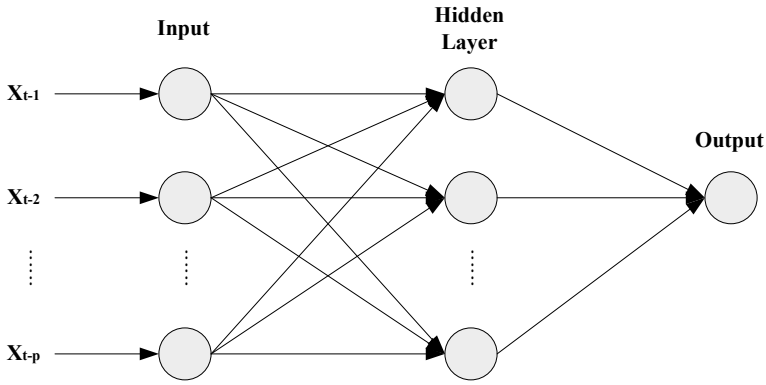

Fig. 1. The multilayer perceptron neural network of a nonlinear AR process

In using the multilayer perceptron neural network to represent (2), one needs to have an input layer with an infinite number of neurons (*infinite memory of the past*), namely, $x_{t-1}, x_{t-2}, ...$, which in practice is impossible. Although, from the viewpoint of approximation, an exact representation is not required and a compromise with a finite number of neurons (*finite memory*) is acceptable, in general quite a few inputs are still required, which inevitably increases the complexity of the network, leads to an unnecessary large number of parameters, and hence slows down the estimation and training process ([38]).

This explains why the multilayer perceptron neural net can only be regarded as the nonlinear extension of autoregressive (AR) time series models

$$x_t = f(x_{t-1}, ..., x_{t-p}) + \epsilon_t, \tag{3}$$

but not the nonlinear extension of the autoregressive moving-average (ARMA) models

$$\begin{aligned} x_t &= f(x_{t-1}, ..., x_{t-p}, \epsilon_{t-1}, ...\epsilon_{t-q}) + \epsilon_t \\ &= f(x_{t-1}, ..., x_{t-p}, x_{t-p-1}, ...) + \epsilon_t \end{aligned} \tag{4}$$

The finite memory problem of the multilayer perceptron neural net is well noticed by ANN researchers. In his celebrated article ([23]), Jeffrey Elman stated

> ...the question of how to represent time in connection models is very important. One approach is to represent time *implicitly* by its effects on processing rather than *explicitly* (as in a spatial representation). (*Ibid*, p.179. Italics added.)

The multilayer perceptron neural net tries to model time by giving it a spatial representation, i.e., an explicit representation. What Elman suggests is to let time have an effect on the network response rather than represent time by an additional input dimension. Using an idea initiated by Michael Jordan ([31]), Elman proposes an internal representation of memory by allowing the hidden unit patterns being to be fed back to themselves. In this way, the network becomes *recurrent*.

The difference between the multilayer perceptron neural network (the feedforward neural network) and the recurrent neural network can be shown as follows. In terms of a multilayer perceptron neural network, (3) can be represented as

$$x_t = h_2(w_0 + \sum_{j=1}^{l} w_j h_1(w_{0j} + \sum_{i=1}^{p} w_{ij} x_{t-i})) + \epsilon_t. \quad (5)$$

Equation (5) is a three-layer neural network (Fig. 1). The input layer has p inputs: $x_{t-1}, ..., x_{t-p}$. The hidden layer has l hidden nodes, and there is a single output for the output layer $\hat{x}_t$. Layers are fully connected by *weights*: w_{ij} is the weight assigned to the ith input for the jth node in the hidden layer, whereas w_j is the weight assigned to the jth node (in the hidden layer) for the output. w_0 and w_{0j} are constants, also called *biases*. h_1 and h_2 are *transfer functions*.

In terms of a recurrent neural network, (4) can then be represented as

$$x_t = h_2(w_0 + \sum_{j=1}^{l} w_j h_1(w_{0j} + \sum_{i=1}^{p} w_{ij} x_{t-i} + \sum_{m=1}^{l} \varpi_{mj} z_{m,t-1})) + \epsilon_t, \quad (6)$$

where

$$z_{m,t} = w_{0m} + \sum_{i=1}^{p} w_{im} x_{t-i} + \sum_{k=1}^{l} \varpi_{kj} z_{k,t-1}, \;\; m = 1, ..., l. \quad (7)$$

In the recurrent neural network, positive feedback is used to construct memory in the network as shown in Fig. 2. Special units called *context units* save previous output values of hidden layer neurons (Eq. 7). Context unit values are then fed back fully connected to hidden layer neurons and serve as additional inputs in the network (Eq. 6).

Compared to the multilayer perceptron neural network and the radial basis function neural network, the recurrent neural network is much less explored in the economic and financial domain.[6] This is, indeed, a little surprising, considering the great exposure of its linear counterpart ARMA to economists.

Chapter 3, **Forecasting Agricultural Commodity Prices using Hybrid Neural Networks**, authored by Tamer Shahwan and Martin Odening, uses recurrent neural networks to forecast the prices of hogs and canolas. The performances of recurrent neural networks are compared with those of ARIMA models, which are frequently used as the benchmark for time series prediction competitions. The authors consider two kinds of recurrent neural networks: the one which works alone, and the one which is hybridized with the ARIMA model. Two empirical issues are, therefore, addressed in this chapter: first, whether the recurrent neural network can outperform the ARIMA model; second, whether the hybrid model can make a further improvement.

The idea of hybridizing the two models is very similar to the familiar two-stage least squares method. In the first stage, the ARIMA model serves as a filter to filter out the linear signal. The residuals are then used to feed the recurrent neural network in the second stage.

[6] Some early applications can be found in [35] and [9].

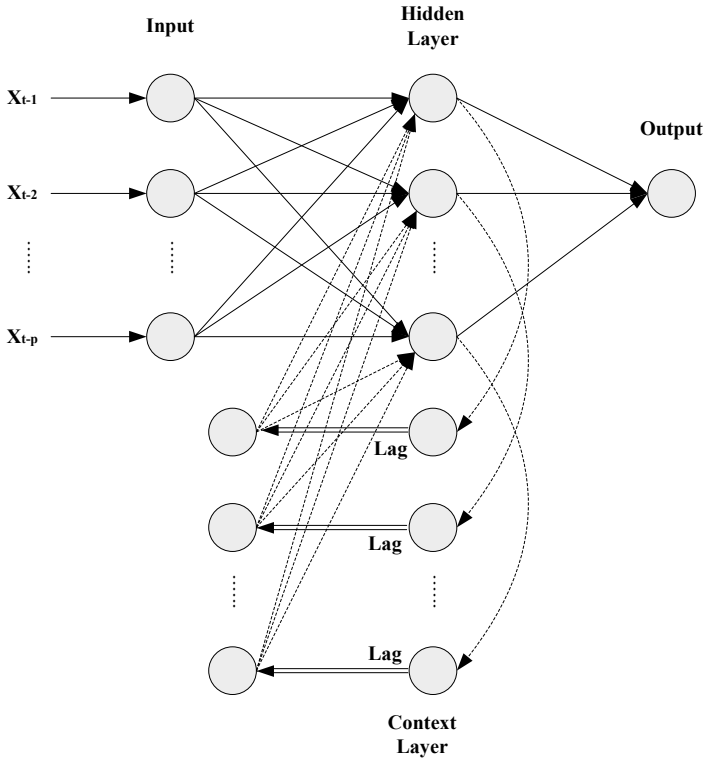

Fig. 2. The recurrent neural network of a nonlinear ARMA process

4.2 Auto-associative Neural Networks

While most economic and financial applications of the neural network consider its capability to develop non-linear forecasting models, as seen in Chap. 3 of the volume, there is one important branch using artificial neural networks to engage in dimension reduction or feature extraction. In this application, ANN can provide a nonlinear generalization of the conventional *principal components analysis* (**PCA**). The specific kind of ANN for this application is referred to as the *auto-associative neural network* (**AANN**).

The fundamental idea of principal components analysis is dimensional reduction, which is a quite general problem when we are presented with a large number of correlated attributes, and hence a large number of redundancies. It is, therefore, a natural attempt to compress or store this original large dataset into a more economical space by getting rid of these redundancies. So, on the one hand, we want to have a reduced space that is as small as possible; on the other hand, we still want to keep the original information. These two objectives are, however, in conflict when attributes with complicated relations are presented. Therefore, techniques to make the least compromise between the two become important.

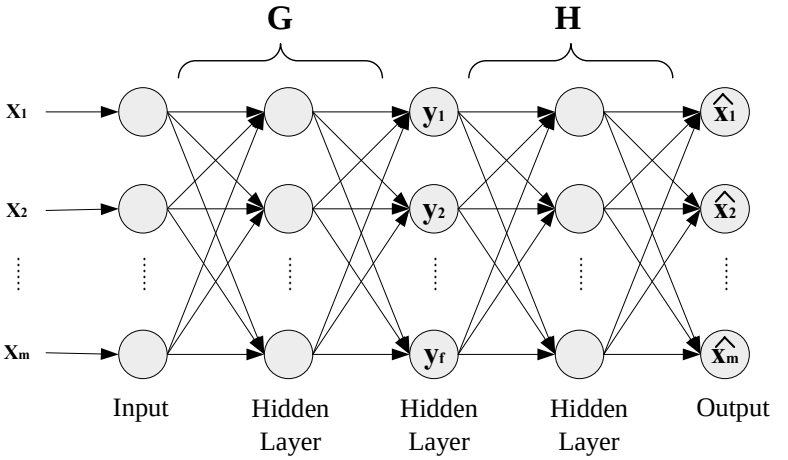

Fig. 3. The Auto-associative Neural Networks

To introduce AANN and its relationship with principal components analysis, let us consider the following two mappings,

$$\mathcal{G} : \mathbf{R}^m \to \mathbf{R}^f \tag{8}$$

and

$$\mathcal{H} : \mathbf{R}^f \to \mathbf{R}^m \tag{9}$$

where $\mathcal{G}$ and $\mathcal{H}$ are, in general, non-linear vector functions with the components indicated as $\mathcal{G} = \{G_1, G_2, ..., G_f\}$ and $\mathcal{H} = \{H_1, H_2, ..., H_m\}$. To represent these functions with multilayer perceptron neural nets, let us rewrite (5) as follows,

$$\begin{aligned} y_k &= G_k(x_1, ..., x_m) \\ &= h_2(w_{0k} + \sum_{j=1}^{l_1} w_{jk} h_1(w_{0j}^e + \sum_{i=1}^{m} w_{ij}^e x_i)), \quad k = 1, 2, ..., f, \end{aligned} \tag{10}$$

and

$$\begin{aligned} \hat{x}_i &= H_i(y_1, ..., y_f) \\ &= h_4(w_{0i} + \sum_{j=1}^{l_2} w_{ji} h_3(w_{0j}^d + \sum_{k=1}^{f} w_{kj}^d y_k)), \quad i = 1, 2, ..., m. \end{aligned} \tag{11}$$

All the notations used in (10) and (11) share the same interpretation as those in (5), except superscripts e and d standing for the encoding and decoding maps, respectively. By combining the two mappings together, we have a mapping from $\mathbf{X} = \{x_1, ..., x_m\}$ to its own reconstruction $\hat{\mathbf{X}} = \{\hat{x}_1, ..., \hat{x}_m\}$. Let X_n be the nth observation of X, and

$$\mathbf{X}_n = \{x_{n,1}, ..., x_{n,m}\}.$$

Accordingly,

$$\hat{\mathbf{X}}_n = \{\hat{x}_{n,1}, ..., \hat{x}_{n,m}\}.$$

Then minimizing the difference between the observation $\mathbf{X}_n$ and its reconstruction $\hat{\mathbf{X}}_n$ over the entire set of N observations or

$$\min E = \sum_{n=1}^{N}\sum_{i=1}^{m}(x_{n,i} - \hat{x}_{n,i})^2 \tag{12}$$

by searching for the space of the connection weights and biases defines what is known as auto-association neural networks. Briefly, auto-associative neural networks are feedforward nets, with *three hidden layers*, trained to produce an approximation of the *identity mapping* between network inputs and outputs using backpropagation or similar learning procedures(See Fig. 3).

The third hidden layer, i.e., the output layer of the MLPN, (10), is also called the *bottleneck layer*. If the transfer functions h_i $(i = 1, 2, 3, 4)$ are all identical mappings, and we remove all the bias terms, then (10) can be written as

$$\begin{aligned} y_k &= G_k(x_1, ..., x_m) \\ &= \sum_{j=1}^{l_1} w_{jk}(\sum_{i=1}^{m} w_{ij}^e x_i) = \sum_{j=1}^{l_1}\sum_{i=1}^{m} w_{jk} w_{ij}^e x_i, \\ &= \sum_{i=1}^{m}\sum_{j=1}^{l_1} w_{jk} w_{ij}^e x_i, = \sum_{i=1}^{m} \beta_{i,k} x_i \;\; k = 1, 2, ..., f, \end{aligned} \tag{13}$$

where

$$\beta_{i,k} = \sum_{j=1}^{l_1} w_{jk} w_{ij}^e$$

In the matrix notations, (13) can be written as

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{bmatrix} \begin{bmatrix} \beta_{11} & \beta_{12} & \dots & \beta_{1f} \\ \beta_{21} & \beta_{22} & \dots & \beta_{2f} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{m1} & \beta_{m2} & \dots & \beta_{mf} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & \dots & y_{1f} \\ y_{21} & y_{22} & \dots & y_{2f} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \dots & y_{nf} \end{bmatrix}, \tag{14}$$

or simply

$$\mathbf{XB} = \mathbf{Y}. \tag{15}$$

$\mathbf{X}$, $\mathbf{B}$ and $\mathbf{Y}$ correspond to the n-by-m, m-by-f, and n-by-f matrices in (14), respectively. Likewise, (11) can be simplified as

$$\mathbf{YB}^* = \hat{\mathbf{X}}. \tag{16}$$

$\mathbf{B}^*$ is the reconstruction mapping and is an f-by-m matrix, and $\hat{\mathbf{X}}$ is the reconstruction of $\mathbf{X}$, and hence is an n-by-m matrix.

Equations (15) and (16) with the objective function (12) define the familiar *linear* principal components analysis. To see this, we can decompose $\mathbf{X}$ as follows:

$$\mathbf{X} = \mathbf{YB}^* + \mathbf{E} = \mathbf{XBB}^* + \mathbf{E} = \mathbf{XP} + \mathbf{E}, \tag{17}$$

where $\mathbf{P} = \mathbf{BB}^*$, and $\mathbf{E}$ is the reconstruction error. Then the PCA frequently presented to us takes the form of the following minimization problem.

$$\min_{\mathbf{P}} || \mathbf{E} || \tag{18}$$

It is known that the optimal solution of to problem (18) has the rows of $\mathbf{P}$ being the eigenvectors corresponding to the f largest eigenvalues of the covariance matrix of $\mathbf{X}$. Therefore, we have shown how the self-associative neural network can be a non-linear generalization of the familiar linear PCA and how the linear PCA can be extended to the non-linear PCA through a feedforward neural network with three hidden layers.

The concept of using a neural network with a bottleneck to concentrate information has been previously discussed in the context of *encoder/decoder* problems. [7] Reference [39] indicates some directions of the financial applications of the nonlinear PCA. In this volume, Chap. 4, **Nonlinear Principal Components Analysis for the Withdrawal of the Employment Time Guarantee Fund**, by Weigang Li, Aipore Rodrigues de Moraes, Lihua Shi, and Raul Yukihiro Matsushita, applies nonlinear principal components analysis to compress a dataset related to employees' withdrawals from a national pension fund in Brazil. It shows how the work on the principal components analysis can be facilitated by the use of artificial neural networks. This chapter provides a good starting point for those who want to see the contribution of artificial neural networks to components analysis. The software mentioned in the paper can be particularly helpful for researchers who want to tackle similar problems of their own.

4.3 Support Vector Machines

The support vector machine (SVM) was introduced in the previous volume ([18], pp.18–20). Two chapters there provide illustrations on the applications of the support vector machine to classifications ([42]) and time series forecasting ([8]). [8] In this volume, Chap. 5, **Estimating Female Labor Force Participation through Statistical and Machine Learning Methods: A Comparison**, authored by Omar Zambrano, Claudio M. Rocco, and Marco Muselli, the SVM is applied to forecast the participation of the female labor force. In this application, the SVM is formally placed in a competitive environment with the conventional linear classification models, i.e., the logit and probit models. In addition, the authors also include a new classification method, referred to as the Hamming clustering, to the competition. They then address the advantages and disadvantages of each approach.

[7] See [34] for a brief review.

[8] Since the publication of the previous volume, the financial applications have kept on expanding, and the interested reader can find some useful references directly from the website of the SVM: http://www.svms.org

4.4 Self-Organizing Maps and K-Means

The genetic programming approach to pattern discovery, as mentioned in Sect. 5.2 below, is a symbolic approach. This approach can also be carried out in different ways by other CI tools, such as decision trees.[9] The symbolic approach makes the patterns or rules discovered explicit in a symbolical way, while, semantically, this is not necessarily so.[10]

However, not all patterns can be expressively demonstrated with symbols. There are other interesting classes of patterns, which can best be visualized as images, trajectories or charts. We see two demonstrations in the first volume. One is [14] which automatically discovers 36 charts by means of the self-organizing map (SOM). In this case, SOM functions as clustering to cluster similar trajectories into the same cluster (cell). The other is [25] which applies K-nearest neighbors (KNN) to choose similar trajectories of time series of exchange rates, and based on those forecasts the future exchange rates. In this volume, we will see the related work of the two. In this section, we introduce the one related to the self-organizing map, i.e., *K-means clustering*, and in the next section, we introduce the one related to K-nearest neighbors, i.e., *instance-based learning*.

K-Means clustering, developed by J.B. MacQueen in 1967 ([37]), is one of the widely used *non-hierarchical clustering algorithms* that groups data with similar characteristics or features together. K-means and SOMs resemble each other. They both involve minimizing some measure of dissimilarity, called the cost function, in the samples within each cluster. The difference between the K-means and the SOM lies in their associated cost function to which we now turn. Consider a series of n observations, each of which has m numeric attributes:

$$\mathbf{X}_1^m, \mathbf{X}_2^m, ..., \mathbf{X}_n^m, \;\; \mathbf{X}_i^m \in \mathbf{R}^m, \;\; \forall\, i = 1, 2, ..., n \tag{19}$$

where

$$\mathbf{X}_i^m \equiv \{x_{i,1}, x_{i,2}, ..., x_{i,m}\}. \;\; x_{i,l} \in \mathbf{R}, \forall\, l = 1, 2, ..., m. \tag{20}$$

K-means clustering means to find a series of k clusters, the centroids of which are denoted, respectively, by

$$\mathbf{M}_1, \mathbf{M}_2, ..., \mathbf{M}_k, \;\; \mathbf{M}_j \in \mathbf{R}^m, \;\; \forall j = 1, 2, ..., k, \tag{21}$$

such that each of the observations is assigned to one and only one of the clusters with a minimal cost, and the cost function is defined as follows:

$$C_{K-means} = \sum_{i=1}^{n} \sum_{j=1}^{k} d(\mathbf{X}_i^m, \mathbf{M}_j) \cdot \delta_{i,j}, \tag{22}$$

[9] In the first volume, we had a detailed discussion on the use of decision trees in finance. See [18], Sect. 1.3.5 and Chap. 15.

[10] What usually happens is that even experts may sometimes find it difficult to make sense of the discovered patterns, and hence it is not certain whether these discovered patterns are spurious. See [19] for a through discussion of the rules discovered by genetic programming. Also see [33] for some related discussion on this issue.

where $d(\mathbf{X}_i^m, \mathbf{M}_j)$ is the standard Euclidean distance between $\mathbf{X}_i^m$ and $\mathbf{M}_j$[11], and $\delta_{i,j}$ is the delta function:

$$\delta_{i,j} = \begin{cases} 1, \, if \;\; \mathbf{X}_i^m \in \mathbf{Cluster}_j, \\ 0, \, if \;\; \mathbf{X}_i^m \notin \mathbf{Cluster}_j. \end{cases} \tag{23}$$

To minimize the cost function (22), one can begin by initializing a set of k cluster centroids. The positions of these centroids are then adjusted iteratively by first assigning the data samples to the nearest clusters and then recomputing the centroids. The details can be found in Chap. 7, **Trading Strategies Based on K-means Clustering and Regression Models**, written by Hongxing He, Jie Chen, Huidong Jin, and Shu-Heng Chen.

Corresponding to (22), the cost function associated with SOM can be roughly treated as follows[12]

$$C_{SOM} = \sum_{i=1}^{n} \sum_{j=1}^{k} d(\mathbf{X}_i^m, \mathbf{M}_j) \cdot h_{w(\mathbf{X}_i^m),j}, \tag{24}$$

where $h_{w(\mathbf{X}_i^m),j}$ is the neighborhood function or the neighborhood kernel, and $w_{\mathbf{X}_i^m}$, the winner function, outputs the cluster whose centroid is nearest to input $\mathbf{X}_i^m$. In practice, the neighborhood kernel is chosen to be wide at the beginning of the learning process to guarantee the global ordering of the map, and both its width and height decrease slowly during learning. For example, the Gaussian kernel whose variance monotonically decreases with iteration times t is frequently used.[13] By comparing Eq. (22) with (24), one can see in SOM the distance of each input from all of the centroids weighted by the neighborhood kernel h, instead of just the closest one being taken into account.

In this volume, Chap. 6 and Chap. 7 are devoted to SOM and KNN, respectively. Chapter 6, **An Application of Kohonen's SOFM to the Management of Benchmarking Policies** authored by Raquel Florez-Lopez, can be read as a continuation of Chap. 9 ([26]) of the first volume. In terms of the research question, it is even closely related to [27]. The core of economic theory of firms is to identify the features of the productivity, efficiency, competitiveness or survivability of firms, or more generally, to answer what makes some firms thrive and others decline. Using observations of firms, economic theory provides different approaches to the answer. Some are more theoretical and require rigid assumptions, while others do not. Data envelopment analysis ([20]) and stochastic frontier analysis ([36]) belong to the former, whereas self-organizing maps belong to the latter. However, what Florez-Lopez does in this chapter is to combine the two: DEA and SOM.

[11] Standard Euclidean distance assumes that the attributes are normalized and are of equal importance. However, this assumption may not hold in many application domains. In fact, one of the main problems in learning is to determine which are the important features.

[12] The rigorous mathematical treatment of the SOM algorithm is extremely difficult in general. See [32].

[13] For details, see the first volume ([18]), Chap. 8, p. 205.

Using DEA, one can distinguish those firms that are on the efficient frontier from those that are not. Nonetheless, without a visualization tool, it is hard to see how similar or different these firms are, be they efficient or inefficient. For example, it is almost impossible by using the conventional DEA to see whether efficient firms are uniformly distributed on the efficient frontier, or whether they are grouped into a few clusters. Hence, it is hard to answer how many viable strategies are available in the market. Using SOM, not only can one see the feature distribution of the efficient firms, but one can also notice how distant, and in what direction, those inefficient firms are from them.

Despite its greater simplicity, the economic and financial applications of K-means are surprisingly much less available than those of SOM and KNN. K-means have occasionally been applied to classify hedge funds ([21]), listed companies ([41]), and houses ([30]), and in this volume, He et al., in Chap. 7, apply them to the classification of trajectories of financial time series. To see this, we rewrite (19) and (20) to fit the notations used in the context of time series:

$$\mathbf{X}_1^m, \mathbf{X}_2^m, ..., \mathbf{X}_T^m, \;\; \mathbf{X}_t^m \in \mathbf{R}^m, \;\; \forall\, t = 1, 2, ..., T \tag{25}$$

$$\mathbf{X}_t^m \equiv \{x_t, x_{t-1}, ..., x_{t-m}\}, \;\; x_{t-l} \in \mathbf{R}, \forall\, l = 0, 1, ..., m-1. \tag{26}$$

$\mathbf{X}_t^m$ is a windowed series with an immediate past of m observations, also called the m-history. Equation (25), therefore, represents a sequence of T m-histories which are derived from the original time series, $\{x_t\}_{t=-m+1}^{T}$, by moving the m-long window consecutively, each with one step. Accordingly, the end-product of applying K-means or SOMs to these windowed series is a number of centroids $\mathbf{M}_j$, which represents a specific shape of an m-long trajectory, also known as charts for technical analysts.[14]

Then the essential question pursued by Chap. 7, as a continuation of Chap. 8 ([14]) in the first volume, is whether we can meaningfully cluster the windowed financial time series $\mathbf{X}_t^m$ by the k associated geometrical trajectories, $\mathbf{M}_1, \mathbf{M}_2, ..., \mathbf{M}_k$. The clustering work can be meaningful if it can help us predict the future. In other words, conditional on a specific trajectory, we can predict the future better than without being provided this information, e.g.,

$$Prob(\mid \xi_{t+1} \mid > \mid \epsilon_{t+1} \mid) > 0.5$$

where

$$\xi_{t+1} = x_{t+1} - E(x_{t+1}), \tag{27}$$

and

$$\epsilon_{t+1} = x_{t+1} - E(x_{t+1} | \mathbf{X_t^m} \in \mathbf{Cluster_j}), \;\; t > T. \tag{28}$$

The conditional expectations above are made with the information of the trajectory (the cluster). Reference [14], in the first volume, is the first one to give this idea a test. They used self-organizing maps to first cluster the windowed time series of the stock index into different clusters, by using the historical data to learn whether these clusters reveal any information, in particular, the future trend of the price. In Chap. 7, the same attempt is carried out again, but now by using K-means clustering.

[14] For example, see the charts presented in [14], pp. 206-207.

4.5 K Nearest Neighbors and Instance-Based Learning

In the first volume, we introduce the financial applications of K-nearest neighbors (KNN) [25]. KNN can be regarded as a special case of a broader class of algorithms, known as *instance-based learning* (IBL). To see this, let us use the notations introduced in Sect. 4.4, and use the time series prediction problem as an illustration.

Consider (28). We have been given information regarding a time series up to time t, and we wish to forecast the next by using the current m-history, $\mathbf{X}_t^m$. In SOM or KNN, we will first decide to which cluster $\mathbf{X}_t^m$ belongs by checking $d(\mathbf{X}_t^m, \mathbf{M}_j)$ for all j ($j = 1, 2, ..., k$), and use the forecast model associated with that cluster to forecast x_{t+1}. In other words, forecasting models are tailored to each cluster, say, $\hat{f}_j$ ($j = 1, 2, ..., k$).[15] Then

$$\hat{x}_{t+1} = \hat{f}_{j^*}(\mathbf{X}_t^m), \;\; if \;\; j^* = \arg\min_j d(\mathbf{X}_t^m, \mathbf{M}_j), \;\; j = 1, 2, ..., k. \tag{29}$$

KNN, however, does not have such established clusters $\mathbf{M}_j$. Instead, it forms a cluster based on each $\mathbf{X}_t^m$, $\mathcal{N}(\mathbf{X}_t^m)$, as follows:

$$\mathcal{N}(\mathbf{X}_t^m) = \{s \mid Rank(d(\mathbf{X}_t^m, \mathbf{X}_s^m)) \leq k, \forall s < t\}, \tag{30}$$

In other words, $\mathbf{X}_t^m$ itself serves as the centroid of a cluster, called the *neighborhood* of $\mathbf{X}_t^m$, $\mathcal{N}(\mathbf{X}_t^m)$. It then invites its k *nearest neighbors* to be the members of $\mathcal{N}(\mathbf{X}_t^m)$ by ranking the distance $d(\mathbf{X}_t^m, \mathbf{X}_s^m)$ over the entire community

$$\{\mathbf{X}_s^m \mid s < t\} \tag{31}$$

from the closest to the farthest. Then, by assuming a functional relation, f, between x_{s+1} and $\mathbf{X}_s^m$ and using only the observations associated with $\mathcal{N}(\mathbf{X}_t^m)$ to estimate this function f_t,[16] one can construct the tailor-made forecast for each x_t,

$$\hat{x}_{t+1} = \hat{f}_t(\mathbf{X}_t^m). \tag{32}$$

In practice, the function f used in (32) can be very simple, either taking the *unconditional mean* or the *conditional mean*. In the case of the latter, the mean is usually assumed to be linear. In the case of the unconditional mean, one can simply use the simple average in the forecast,

$$\hat{x}_{t+1} = \frac{\sum_{s \in \mathcal{N}(\mathbf{X}_t^m)} x_{s+1}}{k}, \tag{33}$$

[15] The notation $\hat{f}$ is used, instead of f, to reserve f for the true relation, if it exists, and in that case, $\hat{f}$ is the estimation of f. In addition, there are variations when constructing (29). See [14] and Chap. 15 in this volume.

[16] Even though the functional form is the same, the coefficients can vary depending on $\mathbf{X}_t^m$ and its resultant $\mathcal{N}(\mathbf{X}_t^m)$. So, we add a subscript t as f_t to make this time-variant property clear.

but one can also take the weighted average based on the distance of each member. The same idea can be applied to deal with the linear conditional mean (linear regression model): we can either take the ordinal least squares or the weighted least squares.[17]

From the above description, we can find that KNN is different from K-means and SOM in the sense that, not just the forecasting function, but also the cluster for KNN is tailor-made. This style of tailor-made learning is known as *lazy learning* in the literature ([2]). It is called *lazy* because learning takes place when the time comes to classify a new instance, say $\mathbf{X}_{T+t}^{m}$, rather than when the *training set*, (25), *is processed*, say T.[18]

To make this clear, consider two types of agents: the K-means agent and the KNN agent. The K-means agent learns from the history before new instances come, and the resultant knowledge from learning is represented by a set of clusters, which is *extracted* from a set of historical instances. Based on these clusters, some *generalization pictures* are already produced before the advent of new instances, say $\mathbf{X}_{T+t}^{m}$.[19] The KNN agent, however, is not eager to learn. While he does store every instance observed, he never tries to extract knowledge (general rules) from them. In other words, he has the simplest form of "learning," i.e., rote learning (plain memorization). When the time $T + t$ comes and a new instance $\mathbf{X}_{T+t}^{m}$ is encountered, his memory is then searched for the historical instances that most strongly resemble $\mathbf{X}_{T+t}^{m}$.

As said, KNN, as a style of rote learning, stores all the historical instances, as shown in (31). Therefore, amounts of storage increase with time. This may make the nearest-neighbor calculation unbearably slow. In addition, some instances may be regarded as redundant with regard to the information gained. This can be particularly the case when KNN is applied to *classification* rather than regression or time series forecasting. For example, if we are interested in not x_{t+1} itself, but in whether x_{t+1} will be greater than x_t, i.e., whether x_t will go up or go down, then some regions of the instance space may be very stable with regard to class, e.g., up (1) or down (0), and just a few exemplars are needed inside stable regions. In other words, we do not have to keep all historical instances or training instances. The *storage-reduction algorithm* is then used to decide which instances in (31) to save and which to discard. This KNN with the storage-reduction algorithm is called *instance-based learning* (IBL) and is initiated by [3].[20]

[17] All details can be found in [25].

[18] Note that a fixed T in (25) implies a fixed training set without increments. A non-incremental training set can be typical for using K-means or SOM. However, KNN learning, also known as *rote learning*, memorizes everything that happens up to the present; therefore, the "training set" (memory) for KNN grows with time.

[19] For example, see Chap. 7 of this volume.

[20] As a matter of fact, the storage-reduction algorithms are not just to deal with the *redundancy* issue, but also the *noise-tolerance* issue. Reference [3] distinguishes the two by calling the former *memory updating functions*, and the latter *noise-tolerant algorithms*. The details can also be found in Chap. 9 of this volume.

The addition of a storage-reduction algorithm to KNN is also interesting from the perspectives of both neural sciences and economics. Considering the brain with its limited capacity for memory, we find that an essential question to ask is how the brain deals with increasing information by not memorizing all of it or by forgetting some of it. How does it do pruning? This is still a non-trivial issue pursued by neural scientists today. The same issue can interest economists as well, because it concerns the efficient use of limited space. A recent study on reward-motivated memory formation by neural scientists may provide an economic foundation for the memory formation ([1]).[21]

In this vein, the *marginal productivity* of the new instance in IBL can be considered as the reward. The marginal productivity of an instance can be defined by its contribution to enhance the capability to perform a correct classification. For those instances which have low marginal productivity, it will be discarded (not be remembered), and for those already stored instances, if their classification performances are poor, they will be discarded, too (be forgotten). In this way, one can interpret the mechanism of the pruning algorithms or the storage-reduction algorithms used in computational intelligence in the fashion of neural economics.

There are two chapters devoted to the financial applications of KNN and IBL. Chapter 8, **Comparison of Instance-Based Techniques for Learning to Predict Changes in Stock Prices**, authored by David LeRoux, uses publicly available monthly economic index data from the Federal Reserve to forecast changes in the S&P 500 stock index. This chapter effectively summarizes a number of technical issues arising from using KNN, including similarity metrics, feature selection, data normalization, choice of the number of neighbors, distance-based weights assigned to the neighbors, and cross validation. The author addresses well the consequence of choosing too many or too few neighbors and involving too many features by demonstrating empirically the impact of different choices of these parameters on accuracy performance.

In addition, the paper serves as a tutorial on a "how-to" guide on using the data mining software known as **WEKA** ([46]).[22] WEKA, along with the book ([46]) actually provides the beginning readers of CIEF with a good starting point to try something on their own.

Chapter 9, **Application of an Instance Based Learning Algorithm for Predicting the Stock Market Index**, authored by Ruppa K. Thulasiram and Adenike Y. Bamgbade, is very similar to Chap. 8 except that it uses IBL, instead of KNN, for predicting the price changes in the S&P 500 daily stock index.

[21] Reference [1] reports brain-scanning studies in humans that reveal how specific *reward-related brain regions* trigger the brain's learning and memory regions to promote memory formation.

[22] WEKA is a collection of machine learning algorithms for solving real-world data mining problems. It is written in Java and runs on almost any platform. The algorithms can either be applied directly to a dataset or called from the users' own Java code.

5 Evolutionary Computation

5.1 Genetic Algorithms

A general introduction to evolutionary computation is given in the previous volume ([18], pp. 33-39). Four branches of evolutionary computation are discussed there. They are genetic algorithms, genetic programming, evolutionary strategies, and evolutionary programming. Reference [15] provides a bibliography of the uses of evolutionary computation in economics and finance. Among the four, the genetic algorithm is the most popular one used in economics and finance.

Chapter 10, **Evaluating the Efficiency of Index Fund Selections over the Fund's Future Period**, authored by Yukiko Orito, Manabu Takeda, Kiyoaki Iimura, and Genji Yamazaki, applies genetic algorithms to the composition of an index fund. The index fund describes a type of mutual fund whose investment objective typically is to achieve the same return as a particular market index, such as Nikkei 255. An index fund will attempt to achieve its investment objective primarily by investing in the securities (stocks or bonds) of companies that are included in a selected index. Some index funds invest in all of the companies included in an index; other index funds invest in a representative sample of the companies included in an index. This paper adopts the second approach, and applies the genetic algorithm to optimize the sample structure, i.e., the portfolio of the representative sample.

5.2 Genetic Programming

Among all kinds of "intelligence" applied to finance, genetic programming is probably the one which requires the most intensive efforts on programming, and its execution is very time-consuming. Therefore, the applications of GP to finance are relatively sparse.[23] In the first volume, [16] provides a review on the financial applications of GP, and, in this volume, two more chapters are added to move the research frontier forward.

In Chap. 11, **The Failure of Computational-Intelligence Induced Trading Strategies: Distinguishing between Efficient Markets and Inefficient Algorithms**, Shu-Heng Chen and Nicolas Navet consider a very fundamental issue: when GP fails to discover profitable trading strategies, how should we react? In the literature, generally, there are two responses: the market is efficient or GP is inefficient. However, there is little clue about which case may be more likely. The two responses can lead to quite different decisions: one is to give up any further attempts on GP, and the other is to propose modified, or even more advanced, versions of GP.

In this chapter, the authors propose a test (a pretest) to help decide which case may be more likely. The test is based upon a very simple idea, namely, comparing the performance of GP with a good choice of benchmark. The authors argued clearly that a good benchmark is not "Buy and Hold", but either random trading strategies or random trading behavior. It involves using these random benchmarks to decide how much we have advanced, and then deciding which case we should refer to.

[23] Up to the present, [10] is the only edited volume devoted to this subject.

The idea of using random benchmarks is not new, and it has been used in developing econometric tests of the predictability of financial time series, which is also well explained in their chapter. Nonetheless, how to define random trading strategies (or trading behavior) is less evident than random time series. Therefore, the two contributions of this chapter are to build these random benchmarks upon a technically acceptable ground, and, as the second part of this chapter, to propose pretests associated with these random benchmarks.

The performance of these pretests is evaluated in light of the earlier results obtained in [19].[24] It is found that when the pretest shows that there is something to learn, GP always performs well; whereas when there is little to learn, GP, as anticipated, accordingly performs rather poorly. Therefore, there is no strong evidence to show that the simple GP is ineffective. It is market efficiency which fails GP. However, the pretests also show that the market is not always efficient; this property changes over time. Therefore, it should not discourage the further use of GP in the financial domain. Basically, when it should work, it works as expected.

As to the flavor which we get from the previous chapter, the earlier applications of financial GP mostly focus on *returns*. While there are some applications which also address risk or volatility, GP has been rarely applied to design a trading strategy which can protect investors against falling stock prices, the so-called *downside risk*.

In Chap. 12, **Nonlinear Goal-Directed CPPI Strategy**, Jiah-Shing Chen and Benjamin Penyang Liao, take up a new challenge of the financial application of GP. CPPI, which stands for *constant proportion portfolio insurance*, is a strategy that allows an investor to limit downside risk while retaining some upside potential by maintaining an exposure to risky assets. It has recently become one of the most popular types of cautious investment instruments ([40]).

However, when investors have a targeted return to pursue, the CPPI strategy may adversely reduce their chances of achieving the goal. To handle this problem, one needs to simultaneously take care of two constraints: the floor (downside risk) constraint and the goal constraint. However, the exact mathematical problem corresponding to solving the two constraint issues may be analytically difficult. Therefore, the authors propose a goal-directed CPPI strategy based on the heuristic motivated by mathematical finance ([6]).

The proposed strategy has a nice property: it is piecewise linear, and hence is simple to operate. The property of linearity is actually based on the assumption of Brownian motion, the cornerstone of modern mathematical finance. It is, however, no longer valid when this assumption is violated. The authors consider two possible cases. In the first case, linearity remains, but the slope coefficient (investment in risky assets) becomes unknown. In the second case, even worse, the linear property is destroyed, and investment in risk assets becomes a non-linear function of wealth. The authors propose a solution for each case separately. A genetic algorithm is used to determine the coefficient of the risky portfolio, and genetic programming is used to determine the nonlinearship relation between the risky portfolio and wealth.

[24] In fact, this chapter can be read as a continuation of the systematic study of the trading application of genetic programming conducted in [19].

The investment strategies developed by GA and GP are further tested by using five stocks selected from the Dow Jones Industrial group. It is found that the piecewise non-linear investment strategy developed by GP performs the best, followed by the piecewise linear investment strategy developed by GA. The piecewise linear investment strategy based on the assumption of Brownian motion performs the worst.

6 Agents

As in the previous volume, the book ends with a chapter on agents. Computational intelligence serves as a foundation for algorithmic agents. By using CI tools, one can then endow agents with capability to learn and to adapt. As to how learning actually takes place, it is detailed by the respective CI tool in an algorithmic manner. These algorithmic agents are then placed in a social (interacting) environment, and the aggregate phenomena are then generated by the collective behavior of these algorithmic agents. It then becomes an experimental study to see how the choice of these algorithms may impact the observed aggregate dynamics ([17]). Based on the sensitivity outcome, one can explore the potential richness of a model, and hence gauge the degree of the inherent uncertainty in the model.

Based on what we have discussed in the previous sections, there are many ways of comparing different learning algorithms when applied to building algorithmic agents. Reference [5], for example, from a viewpoint of cognitive psychology provides a way of comparing some frequently used learning algorithms in economics. [25] Cognitive loading is certainly an important concern when one wants to choose an appropriate learning algorithm to model boundedly rational agents. A highly relevant question to ask is how demanding these algorithms are in terms of cognitive loading, and whether there is a threshold beyond which the agents' "humble mind" can simply not afford those demanding tasks. Answers to these questions are more subtle than one may think at first sight.

Reference [17] addresses the problem of how to relate and compare agent behavior based on computational intelligence models to human behavior, and proposes the condition of computational equivalence to deal with this difficult issue. The paper describes the design of a computational equivalence lab where both humans and software agents have the same computational intelligence methods at their disposal. In a similar vein, Chap. 13, **Hybrid-Agent Organization Modeling: A Logic-Heuristic Approach**, authored by Ana Marostica, Cesar A. Briano and Ernesto Chinkes, proposes a hybrid-agent organization model by using some ideas from scientific semiotics. This chapter illustrates the idea of integrating human agents (decision makers) with software agents (heuristic decision support system) and the connection between the two.

[25] In addition, these learning algorithms are not just confined to individual agents, i.e., agents only learn from their own experience. In fact, a class of learning algorithms, called *social learning*, explicitly or implicitly assumes the existence of a social network and indicates how agents learn from others' experiences as well. For example, see [13].

7 Concluding Remarks

While this second volume provides us with a good opportunity to include some interesting subjects which are missing in the first volume, these two volumes together are still not large enough to accommodate all state-of-the-art CI techniques with their economic and financial applications. To name a few, reinforcement learning, independent component analysis, and artificial immune systems are techniques that we hope to include in the next volume. In addition, some popular hybrid systems, such as fuzzy C-means, neuro-fuzzy systems, fuzzy-neural networks, genetic fuzzy systems, fuzzy evolutionary algorithms and genetic Bayesian networks are other techniques that will be included in the future as well.

Acknowledgements

The editors would like to acknowledge the efforts made by all referees, including Chao-Hsien Chu, Wei Gao, Steven C. Gustafson, Chao-Fu Hong, Carson Kai-Sang Leung, Tong Li, Jie-Shin Lin, Ping-Chen Lin, Martin Odening, Yukiko Orit, Claudio M. Rocco, Giulia Rotundo, Jishou Ruan, Arnold F. Shapiro, Theodore Theodosopoulos, Ruppa K. Thulasiram, Chueh-Yung Tsao, Chung-Tsen Tsao, Chiu-Che Tseng, Tzai-Der Wang, Weigang Li, Ana Marostica, Fernando Tohme and Raquel Florez-Lopez. To enhance the quality of the volume, all chapters, apart from those that are authored by native speakers, have been sent to an English editor for reviewing. In this regard, we are particularly grateful to Bruce Stewart for his excellent editing service.

References

1. Adcock A, Thangavel A, Whitfield-Gabrieli S, Knutson B, Gabrieli J (2006) Reward-motivated learning: mesolimbic activation precedes memory formation. Neuron 50(3):507–517
2. Aha D (1997) Lazy learning. Kluwer
3. Aha D, Kibler D, Marc K (1991) Instance-based learning algorithms. Machine Learning 6(1):37–66
4. Alvarez-Diaz M, Alvarez A (2005) Genetic multi-model composite forecast for nonlinear prediction of exchange rates. Empirical Economics 30:643–663
5. Brenner T (2006) Agent learning representation: Advice on modelling economic learning. In: Tesfatsion L, Judd K (eds) Handbook of computational economics, volume 2: agent-based computational economics. North-Holland, Amsterdam:895–947
6. Browne S (1997) Survival and growth with a liability: optimal portfolio strategies in continuous time. Mathematics of Operations Research 22(2):468–493
7. Campbell J, Lo A, MacKinlay C (1997) The econometrics of financial markets. Princeton University Press, NJ
8. Cao L, Tay F (2003) Saliency analysis of support vector machines for feature selection in financial time series forecasting. In: Chen SH, Wang P (eds) Computational intelligence in economics and finance. Springer-Verlag:182–199

9. Chen J, Xu D (1998) An economic forecasting system based on recurrent neural networks. In: Proceedings of IEEE international conference on systems, man, and cybernetics:1762–1767
10. Chen SH (ed) (2002) Genetic algorithms and genetic programming in computational finance. Kluwer.
11. Chen SH (2005) Computational intelligence in economics and finance: carrying on the legacy of Herbert Simon. Information Sciences 170:121–131
12. Chen SH (2007a) Computationally intelligent agents in economics and finance. Information Sciences 177(5):1153-1168
13. Chen SH (2007b) Graphs, networks and ACE. New Mathematics and Natural Computation 2(3):299–314
14. Chen SH, He H (2003) Searching financial patterns with self-organizing maps. In Chen SH, Wang P (eds) Computational intelligence in economics and finance. Springer-Verlag:203–216
15. Chen SH, Kuo TW (2002) Evolutionary computation in economics and finance: A bibliography. In: Chen SH (eds) Computational intelligence in economics and finance. Springer-Verlag:419–455
16. Chen SH, Kuo TW (2003) Discovering hidden patterns with genetic programming. In Chen SH, Wang P (eds) Computational intelligence in economics and finance. Springer-Verlag:329-347
17. Chen SH, Tai CC (2007) On the selection of adaptive algorithms in ABM: a computational-equivalence approach. Computational Economics 28:313–331
18. Chen SH, Wang P (eds) (2003) Computational intelligence in economics and finance. Springer-Verlag.
19. Chen SH, Kuo TW, Hoi KM (2007) Genetic programming and financial trading: how much about "What we Know"? In: Zopounidis C, Doumpos M, Pardalos PM (eds) Handbook of financial engineering. Springer. Forthcoming.
20. Cooper W, Seiford L, Tone K (2005) Introduction to data envelopment analysis and its uses: with DEA-Solver software and references. Springer.
21. Das N (2003) Hedge fund classification using K-means method. EconPapers, No. 204
22. De Cock M (1999) Representing the adverb very in fuzzy set theory. ESSLLI Student Papers 1999:223–232
23. Elamn J (1990) Finding structure in time. Cognitive Science 14:179–211
24. Episcopos A, Davis J (1996) Predicting returns on Canadian exchange rates with artificial neural networks and EGARCHM-M models. Neural Computing and Application 4:168–174
25. Fernández-Rodríguez F, Sosvilla-Rivero S., Andrada-Félix J (2003) Nearest-neighbour predictions in foreign exchange markets. In Chen SH, Wang P (eds) Computational intelligence in economics and finance. Springer-Verlag:297–325
26. Florez-Lopez R (2003) Effective position of European firms in the face of monetary integration using Kohonen's SOFM. In Chen SH, Wang P (eds) Computational intelligence in economics and finance. Springer-Verlag:217–233
27. Florez-Lopez R (2007) Strategic supplier selection in the added-value perspective: a CI approach. Information Science 177(5):1169–1179
28. Hann T, Steurer E (1996) Much ado about nothing? Exchange rate forecasting: neural networks vs. linear models using monthly and weekly data. Neurocomputing 10:323–339
29. Hersh H, Caramazza A (1976) A fuzzy set approach to modifiers and vagueness in natural language. Journal of Experimental Psychology–General 105 (3):254–276

30. Hollans H, Munneke HJ (2003) Housing markets and house price appreciation: an Intercity Analysis. Working paper. Univeristy of Georgia.
31. Jordan M (1986) Serial order: a parallel distributed processing approach. Technical Report, No. 8604. Institute for Cognitive Science, UC San Diego
32. Kohonen T (1995) Self-organizing Maps. Springer.
33. Kovalerchuk B, Vityaev E (2000) Data mining in finance: advances in relational and hybrid methods. Kluwer.
34. Kramer M (1990) Nonlinear principal analysis using autoassociative neural networks. AIChE Journal 37(2):233–243
35. Kuan CM, Liu T (1995) Forecasting exchange rates using feedforward and recurrent neural networks. Journal of Applied Econometrics 10:347–364
36. Kumbhakar S, Knox Lovell C (2003) Stochastic frontier analysis. Cambridge.
37. MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press 1:281–297
38. Mandic D, Chambers J (2001) Recurrent neural networks for prediction: learning Algorithms, architectures, and stability. John Wiley and Sons, New York
39. McNelis P (2005) Neural networks in finance: gaining predictive edge in the market. Elsvier.
40. Overhaus M, Bermudez A, Buehler H, Ferraris A, Jordinson C, Lamnouar A (2007) Equity hybrid derivatives. Wiley Finance
41. Qian Y (2006) K-means algorithm and its application for clustering companies listed in Zhejiang province. WIT Transaction on Information and Communication Technologies 37:35–44
42. Rocco C, Moreno J (2003) A support vector machine model for currency crises discrimination. In: Chen SH, Wang P (eds) Computational intelligence in economics and finance. Springer-Verlag:171–181
43. Shi S, Xu L, Liu B (1999) Improving the acuracy of nonlinear combined forecasting using neural networks. Expert Systems with Applications 16:49–54
44. Wei WX, Jiang ZH (1995) Artificial neural network forecasting model for exchange rate and empirical analysis. Forecasting 2:67-V69
45. Weigend A, Huberman B, Rumelhart D (1992) Predicting sunspots and exchange rates with connectionist networks. In: Casdagli M, Eubank S (eds) Nonlinear modeling and forecasting. Addison-Wesley:395V-432
46. Witten I, Frank E (2005) Data mining: practical machine learning tools and techniques (2nd Edition). Morgan Kaufmann
47. Wu B (1995) Model-free forecasting for nonlinear time series (with application to exchange rates). Computational Statistics and Data Analysis 19:433–459