Application of type-2 neuro-fuzzy modeling in stock price prediction

Chih-Feng Liu, Chi-Yuan Yeh, Shie-Jue Lee *  
Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung 804, Taiwan

ABSTRACT

We present an application of type-2 neuro-fuzzy modeling to stock price prediction based on a given set of training data. Type-2 fuzzy rules can be generated automatically by a self-constructing clustering method and the obtained type-2 fuzzy rules can be refined by a hybrid learning algorithm. The given training data set is partitioned into clusters through input-similarity and output-similarity tests, and a type-2 TSK rule is derived from each cluster to form a fuzzy rule base. Then the antecedent and consequent parameters associated with the rules are refined by particle swarm optimization and least squares estimation. Experimental results, obtained by running on several datasets taken from TAIEX and NASDAQ, demonstrate the effectiveness of the type-2 neuro-fuzzy modeling approach in stock price prediction.

1. Introduction

Forecasting of stock prices is an appealing yet difficult activity in the modern business world. Many factors influence the behavior of the stock market, either economic or non-economic. Therefore, stock prediction is regarded as one of the most challenging topics in business. In the past, methods based on statistics were proposed for tackling this problem, such as the autoregressive (AR) model [1], the autoregressive moving average (ARMA) model [2], and the autoregressive integrated moving average (ARIMA) model [2]. These are linear models which are, more than often, inadequate for stock market forecasting. Non-linear approaches were subsequently proposed, such as autoregressive conditional heteroskedasticity (ARCH) [3] and generalized autoregressive conditional heteroskedasticity (GARCH) [4]. Recently, many soft computing approaches have been proposed for stock market price forecasting, including fuzzy time series models [5–7], artificial neural networks [8–17], fuzzy neural networks [18–20], support vector regression machines [21–24], and hybrid forecasting models [25–29].

Huang et al. [5] proposed a multivariate heuristic model which integrates the univariate fuzzy time series model with a multivariate heuristic function for TAIEX forecasting. Daily data taken from NASDAQ (USA) [30], Dow Jones (USA) [31], and M1b (Taiwan) [32] are used as heuristic variables to help predict the fall or down of the TAIEX price in a heuristic way. An m-dimensional heuristic table is defined, where m is the number of heuristic variables. According to the heuristic table, the proper fuzzy sets can be chosen from the right-hand side of the members in the fuzzy logical relationship groups. Yu and Huang [6] proposed a weighted model for forecasting the Taiwan Stock Exchange Capitalization Weighted Stock Index (TAIEX) [33], and claimed its superiority over conventional models. Chen and Chen [7] proposed a forecasting model for TAIEX forecasting based on fuzzy time series and fuzzy variation groups. They also use the Dow Jones and the NASDAQ as secondary factors to improve the forecasting performance of the model. However, fuzzy time series models lack an effective learning algorithm to refine the associated membership functions to minimize output errors.

Yoon et al. [8] applied artificial neural networks (ANNs) to stock price prediction. Experimental results showed that ANNs outperform a linear discriminant analysis (DA) model. Saad et al. [9] applied three ANNs, time-delay, probabilistic, and recurrent neural networks, respectively, to stock price prediction. Kuo et al. [18] proposed a GA-based fuzzy neural network to measure the qualitative effects on the stock market. Huang and Yu [13] proposed neural network-based fuzzy time series models for TAIEX forecasting. The indices of fuzzy sets of a fuzzy logical relationship are used for training the neural network. The proposed models were claimed to outperform many other methods. Kwon and Moon [14] used a hybrid genetic approach combined with recurrent ANNs to predict the stock price. The genetic algorithm (GA) is used to find optimal weights for ANNs. Chang and Liu [19] proposed a

* Corresponding author. Tel.: +886 7 5252000.  
E-mail address: leesj@mail.ee.nsysu.edu.tw (S.-J. Lee).
TSK type neuro-fuzzy system for stock price prediction. Simulated annealing (SA) is used to search optimal parameters for fuzzy rules.

The stock market is a complex, evolutionary, and nonlinear dynamical system. The price variation of stock market is inherently noisy, non-stationary, and deterministically chaotic [34]. A variety of factors, e.g., the statements from politicians or the speeches of corporation executives, can influence the stock price. Most of them are stated in fuzzy linguistic terms. These factors are interpreted differently by the stock buyers and cause different effects on the stock price. The daily closing prices are just real numbers which are precise. However, different numbers might come out from different interpretations. This is where uncertainty or imprecision comes out. The stock market also involves uncertainty which stems from lack of complete information [35–37]. Most proposed fuzzy-based stock price prediction methods adopted type-1 fuzzy sets which represent uncertainties by numbers in the range [0, 1]. The membership functions of type-1 fuzzy sets are often overly precise, requiring each element of the universal set be assigned a particular real number. Type-2 fuzzy sets were proposed for dealing with this difficulty [38]. A type-2 fuzzy set allows its associated membership degrees to be uncertain [39–42] and is more suitable than a type-1 fuzzy in handling uncertainties such as those related to noisy or non-stationary conditions. Zarandi et al. [20] proposed an interval type-2 fuzzy rule-based expert system model for stock price forecasting. The fuzzy Gustafson–Kessel clustering algorithm based on a cluster validity measure is used to determine the initial fuzzy rules. Then GA is used for tuning the parameters of fuzzy rules.

We present an application of type-2 neuro-fuzzy modeling to stock price prediction based on a given set of training data. Type-2 fuzzy rules are generated automatically by a self-constructing clustering method [43] and the obtained type-2 fuzzy rules are refined by a hybrid learning algorithm. The self-constructing clustering method partitions the training data set into clusters through input-similarity and output-similarity tests. The membership function associated with each cluster is defined with the mean and deviation of the data points included in the cluster. A type-2 TSK rule is derived from each cluster to form a fuzzy rule base. Then a hybrid learning algorithm, which incorporates particle swarm optimization and least squares estimation, is used to refine the antecedent parameters and the consequent parameters, respectively, associated with the rules. For a new input, a corresponding crisp output is obtained by combining the inferred results of all the rules into a type-2 fuzzy set which is then defuzzified by applying a refined type reduction algorithm. The effectiveness of the type-2 neuro-fuzzy modeling approach in stock price prediction is tested on data taken from TAIEX and NASDAQ, and compared with that of some other methods using several error measures such as RMSE, MAE, and MAPE [44].

The rest of this paper is organized as follows. Section 2 introduces some basic concepts about type-2 fuzzy sets. Converting a type-2 fuzzy set to a crisp number is described. The derivation of type-2 fuzzy rules from a given set of input–output training data is described in Section 3. Applying learning techniques to refine the parameters associated with the rules is described in Section 4. An illustrating example is given in Section 5. Experimental results are presented in Section 6. Finally, a conclusion is given in Section 7.

2. Basic concepts

In this section, some basic concepts about type-2 fuzzy sets are introduced. Before proceeding, we introduce type-1 fuzzy sets. Given a universal set $X$, a type-1 fuzzy set $A$ is defined by

$$ A = \{(x, \mu_A(x)) \mid \forall x \in X\} $$

where $\mu_A$ is the membership function of $A$:

$$ \mu_A: X \rightarrow [0, 1]. $$

Fig. 1(a) shows an example of type-1 fuzzy sets.

2.1. Type-2 fuzzy sets

A type-2 fuzzy set $\tilde{A}$ on a given universal set $X = \{x_1, x_2, \ldots, x_N\}$, where $x_1 < x_2 < \ldots < x_N$, can be represented as [45]

$$ \tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid \forall x \in X\} $$

in which $\mu_{\tilde{A}}(x)$, called the secondary membership function, refers to a type-1 fuzzy set

$$ \mu_{\tilde{A}}(x) = \int_{u \in \tilde{A}} f_u(u) \, du $$

where $f_u \subseteq [0, 1]$ and $0 \leq f_u(u) \leq 1$. In this paper, we assume that a type-2 fuzzy set $\tilde{A}$ defined on a universal set $X$ has the following secondary membership function form:

$$ \mu_{\tilde{A}}(x) = \text{gauss}(u; \text{gauss}(x; m^p, \sigma^p), \sigma^p) $$

where $x \in X$, $u \in [0, 1]$, and $\text{gauss}(x; m, \sigma)$ is the Guassian function with mean $m$ and deviation $\sigma$ defined as

$$ \text{gauss}(x; m, \sigma) = \exp \left[ -\left( \frac{x - m}{\sigma} \right)^2 \right]. $$

One example of such type-2 fuzzy sets is shown in Fig. 1(b). As in [46], the inner Guassian $\text{gauss}(x; m^p, \sigma^p)$ is called the primary membership function for $\tilde{A}$. For example, suppose we have $m^p = 5$ and $\sigma^p = 2$. Then for $x = 3$, $\text{gauss}(3; 5^p, 2^p) = e^{-1}$. Furthermore, let $\sigma^p = 0.2$. Then we have

$$ \mu_{\tilde{A}}(3) = \text{gauss}(u; \text{gauss}(3; m^p, \sigma^p), \sigma^p) = \exp \left[ -\left( \frac{u - e^{-1}}{0.2} \right)^2 \right] $$

which is a type-1 fuzzy set with $u \in [0, 1]$.

2.2. Crisp value for a type-2 fuzzy set

In a type-2 fuzzy system, we need to find a crisp number as the system output most of the time. In this section, we introduce the techniques of converting a type-2 fuzzy set into a crisp number.

Liu [46] proposed a method to find the centroid of a type-2 fuzzy set. For a type-2 fuzzy set $\tilde{A}$, the centroid of $\tilde{A}$, denoted $C(\tilde{A})$, is defined as

$$ C(\tilde{A}) = \frac{1}{D} \sum_{j=1}^{D} \alpha_j \tilde{C}(\alpha_j) $$

where $D$ is a positive integer and $\alpha_j = j/D$ for $j = 1, \ldots, D$, and $\tilde{C}(\alpha)$ and $\tilde{C}(\alpha)$ are computed by [35,45,47]

$$ \tilde{C}(\alpha_j) = \frac{\sum_{i=1}^{l_{\alpha_j}} x_{l_{\alpha_j}}(x_i) + \sum_{i=l_{\alpha_j}+1}^{N} x_{l_{\alpha_j}}(x_i)}{\sum_{i=1}^{l_{\alpha_j}} I_{l_{\alpha_j}}(x_i) + \sum_{i=l_{\alpha_j}+1}^{N} I_{l_{\alpha_j}}(x_i)}, $$

$$ \tilde{C}(\alpha_j) = \frac{\sum_{i=1}^{R_{\alpha_j}} x_{R_{\alpha_j}}(x_i) + \sum_{i=R_{\alpha_j}+1}^{N} x_{R_{\alpha_j}}(x_i)}{\sum_{i=1}^{R_{\alpha_j}} I_{R_{\alpha_j}}(x_i) + \sum_{i=R_{\alpha_j}+1}^{N} I_{R_{\alpha_j}}(x_i)}. $$

Note that in these equations,

$$ x_{l_{\alpha_j}} \leq \tilde{C}(\alpha_j) < x_{l_{\alpha_j}+1}, $$

$$ x_{R_{\alpha_j}} \leq \tilde{C}(\alpha_j) < x_{R_{\alpha_j}+1}. $$
for \( j = 1, \ldots, D \). We adopt the algorithm proposed in [48], which is an enhancement to Liu’s method, to compute \( C(\hat{A}) \) in Eq. (7). Note that the centroid obtained by Eq. (7) is a type-1 fuzzy set. A crisp value can then be obtained by defuzzifying Eq. (7) as

\[
\text{crisp} = \frac{\int C(\hat{A})dx}{\int C(\hat{A})dx}
\]

\[
= \frac{\sum_{j=1}^{D-1} \int_{\xi_j}^{\xi_{j+1}} \alpha_j dx + \int_{\xi_j}^{\xi_{D}} \alpha_j dx + \sum_{j=2}^{D} \int_{\xi_j}^{\xi_{j-1}} \alpha_j dx}{\sum_{j=1}^{D-1} \int_{\xi_j}^{\xi_{j+1}} \alpha_j dx + \int_{\xi_j}^{\xi_{D}} \alpha_j dx + \sum_{j=2}^{D} \int_{\xi_j}^{\xi_{j-1}} \alpha_j dx}
\]

(13)

which can be approximated by [49]

\[
\text{crisp} \approx \frac{\sum_{j=1}^{D} \alpha_j \cdot (\xi_{j+1} - \xi_j)/2}{\sum_{j=1}^{D} \alpha_j}.
\]

(14)

3. Rule base construction

In this section, we describe how a rule base, consisting of a set of type-2 fuzzy rules, is derived from a given set of training data [50]. Assume the system to be modeled has \( n \) inputs, \( x_1, x_2, \ldots, x_n \) and one output \( y \). Let the given training dataset contain \( \ell \) patterns \((p^{(1)}, d^{(1)}), (p^{(2)}, d^{(2)}), \ldots, (p^{(\ell)}, d^{(\ell)})\)

where \( p^{(r)} = [p_1^{(r)}, p_2^{(r)}, \ldots, p_n^{(r)}] \) denotes the \( n \) input values and \( d^{(r)} \) denotes the corresponding desired output value of the \( r \)th pattern, \( 1 \leq r \leq \ell \). Firstly, a self-constructing clustering method [43] is employed to partition the training data into a collection of clusters, and the clusters are converted into a rule base of type-2 fuzzy TSK IF–THEN rules. Secondly, the parameters associated with the rules are refined through a hybrid learning algorithm to be presented later.

3.1. Finding clusters

The self-constructing clustering method partitions the training dataset into clusters through input-similarity and output-similarity tests. A cluster contains a certain number of training patterns, and is characterized by the product of \( n \) one-dimensional Gaussian functions. Training patterns are considered one by one. At the beginning, no clusters exist, but clusters are added later if necessary. For each pattern, the similarity of this pattern to each existing cluster is calculated to decide whether it is combined into an existing cluster or a new cluster is created. When all training patterns have been considered, we get a number of clusters.

Let \( k \) be the number of currently existing clusters denoted by \( G_1, G_2, \ldots, G_k \), respectively. Each cluster \( G_j \) has mean \( m_j = [m_{1j}, m_{2j}, \ldots, m_{nj}] \), deviation \( \sigma_j = [\sigma_{1j}, \sigma_{2j}, \ldots, \sigma_{nj}] \), and height \( h_j \) which is the average of the desired outputs of the patterns contained in \( G_j \).

Let \( S_j \) be the size of cluster \( G_j \), i.e., the number of training patterns contained in \( G_j \). Initially, we have \( k = 0 \). So, no clusters exist at the beginning. For pattern \( r \), i.e., \((p^{(r)}, d^{(r)})\), \( 1 \leq r \leq \ell \), we calculate the input similarity of pattern \( r \) to each existing cluster \( G_j \) as follows:

\[
\mu_{G_j}(p^{(r)}) = \prod_{i=1}^{n} \exp \left( -\frac{(p_i^{(r)} - m_{ij})^2}{\sigma_{ij}^2} \right)
\]

(15)

for \( 1 \leq j \leq k \). Notice that \( 0 \leq \mu_{G_j}(p^{(r)}) \leq 1 \). In the case of \( \mu_{G_j}(p^{(r)}) \approx 1 \), \( p^{(r)} \) is near the center of cluster \( G_j \) and we consider that \( p^{(r)} \) is highly likely to belong to \( G_j \). On the contrary, if \( \mu_{G_j}(p^{(r)}) \approx 0 \), \( p^{(r)} \) is far away from the center of cluster \( G_j \) and we consider that \( p^{(r)} \) is very unlikely to belong to \( G_j \). We say that pattern \( r \) passes the input similarity test on cluster \( G_j \) if

\[
\mu_{G_j}(p^{(r)}) \geq \rho
\]

(16)

where \( \rho, 0 \leq \rho \leq 1 \), is a predefined threshold. Note that when the threshold increases, it is less likely for a cluster to pass the test. Furthermore, we calculate the output similarity of pattern \( r \) to each existing cluster \( G_j \) as follows:

\[
e_j = |d^{(r)} - h_j|.
\]

(17)

Let \( d_{\max} \) and \( d_{\min} \) denote the maximum and minimum of all the desired outputs, i.e.,

\[
d_{\max} = \max_{1 \leq i \leq \ell} d^{(i)}, \quad d_{\min} = \min_{1 \leq i \leq \ell} d^{(i)}.
\]

We say that pattern \( r \) passes the output-similarity test on cluster \( G_j \) if

\[
e_j \leq \tau (d_{\max} - d_{\min})
\]

(18)

where \( \tau, 0 \leq \tau \leq 1 \), is another predefined threshold. As \( \tau \) decreases, the test gets tougher and more clusters will be produced.

Two cases may occur. Firstly, there are no existing clusters on which pattern \( r \) has passed both the input-similarity test and the output-similarity test. In this case, we assume that pattern \( r \) is not similar enough to any existing cluster. We increase \( k \) by one. A new cluster is created.

\[
(L_j(x_i), T_j(x_i)) = (u \mid u \in F_j \wedge f_k(u) \geq \alpha_j),
\]

(12)
cluster is created by setting its mean to be \( p^{(r)} \) itself, its deviation to be a default vector \( \sigma_0 = [\sigma_0, \ldots, \sigma_0] \), and its height to be \( d^{(r)} \), i.e.,

\[
k = k + 1, \quad m_k = p^{(r)}, \quad \sigma_k = \sigma_0, \quad h_k = d^{(r)}.
\] (19)

Note that the deviation of this new cluster \( G_k \) is 0 since it contains only one member. We cannot use zero deviation in Eq. (15). Therefore, we initialize the deviation \( \sigma_0 \) of \( G_k \) to be \( \sigma_0 \). On the other hand, if there are existing clusters on which pattern \( r \) has passed both the input-similarity test and the output-similarity test, let \( G_{j_1}, G_{j_2}, \ldots, G_{j_r} \) be such clusters and let the cluster with the greatest input similarity be cluster \( i \), i.e.,

\[
t = \arg \max_{1 \leq j \leq r} \mu_{G_j}(p^{(r)}).
\] (20)

In this case, we assign pattern \( r \) cluster \( G_i \). The mean \( m_i \), the deviation \( \sigma_i \), and the height \( h_i \) of cluster \( G_i \) are modified to include pattern \( i \) as its member as follows:

\[
\sigma_i, t = \sqrt{A - B},
\]

\[
A = \frac{(S_t - 1)\sigma_{i,t}^2 + S_t \times m_{i,t}^2 + (p_{i,t}^{(r)})^2}{S_t},
\]

\[
B = \frac{S_t + 1}{S_t} \left( \frac{S_t \times m_{i,t} + p_{i,t}^{(r)}}{S_t + 1} \right)^2,
\]

\[
m_{i,t} = \frac{S_t \times m_{i,t} + p_{i,t}^{(r)}}{S_t + 1},
\]

\[
h_t = \frac{S_t \times h_t + d^{(r)}}{S_t + 1},
\]

for \( 1 \leq i \leq n, \) and

\[
S_t = S_t + 1
\] (25)

which increases the size of \( G_t \) by 1.

Our clustering method possesses a similar idea to dynamic fuzzy clustering [51–53]. For a training pattern, the similarity of this pattern to each existing cluster is calculated to decide whether it is combined into an existing cluster or a new cluster is created. The training process may continue when new training patterns are encountered. However, unlike dynamic fuzzy clustering, our clustering method does not perform updating of initial clusters. This may result in a larger number of clusters, which means that more rules may be obtained. Empirically, such an increase in the number of resulting rules can hardly affect the performance of the obtained rule base.

Note that the order in which the training patterns are fed in influences the clusters obtained. Different orderings may result in different numbers of clusters, leading to different models with different numbers of fuzzy rules. However, the prediction accuracies of the obtained models do not vary significantly.

3.2. Deriving rules

Suppose we have \( J \) clusters when all the training patterns have been considered. We convert each cluster to a type-2 TSK fuzzy rule and have a set of \( J \) rules, \( R_1, R_2, \ldots, R_J \), each having the following form:

\[
R_j : \quad \text{IF } x_1 \text{ IS } \tilde{A}_{1,j} \text{ AND } x_2 \text{ IS } \tilde{A}_{2,j} \text{ AND } \ldots \text{ AND } x_n \text{ IS } \tilde{A}_{n,j} \text{ THEN } y_1 \text{ IS } p^{(1)}_j, y_2 \text{ IS } p^{(2)}_j, \ldots, y_s \text{ IS } p^{(s)}_j
\] (26)

where \( p_{0j}, p_{1j}, \ldots, p_{nj} \) are real-valued parameters and \( \tilde{A}_{1,j}, \tilde{A}_{2,j}, \ldots, \tilde{A}_{n,j} \) are type-2 fuzzy sets for \( x_1, x_2, \ldots, x_n \), respectively. The membership function of each \( \tilde{A}_{i,j} \), \( 1 \leq i \leq n, 1 \leq j \leq J \), takes the form of Eq. (5), i.e.,

\[
\mu_{\tilde{A}_{i,j}}(x_i) = \text{gauss}(u; \text{gauss}(x_i; m_{i,j}^p, \sigma_{i,j}^p), \sigma_{i,j}^s)
\] (27)

where \( u \in [0, 1] \). The antecedent parameters of rule \( R_j \) include \( m_{i,j}^p, \sigma_{i,j}^p, \) and \( \sigma_{i,j}^s, 1 \leq i \leq n, \) and they are initialized as the mean and deviation of cluster \( G_j \) by

\[
m_{i,j}^p = m_{i,j},
\]

\[
\sigma_{i,j}^p = \sigma_{i,j},
\]

\[
\sigma_{i,j}^s = \kappa \sigma_{i,j}
\] (28)

for \( 1 \leq i \leq n \), where \( \kappa \) is a small, user-defined constant, e.g., \( \kappa = 0.1 \). The consequent parameters of rule \( R_j \) include \( \beta_{0j}, \beta_{1j}, \ldots, \beta_{nj} \), and they are initialized by least squares estimation. Let the \( S_t \) training patterns contained in cluster \( G_j \) be \( (p^{(1)}_1, d^{(1)}_1), (p^{(2)}_1, d^{(2)}_1), \ldots, (p^{(K)}_1, d^{(K)}_1) \). Then \( p_{0j}, p_{1j}, \ldots, p_{nj} \) are set to be the least squares solution to the following linear system

\[
X \beta = Y
\] (31)

where

\[
X = \begin{bmatrix} 1 & p^{(1)}_1 & p^{(2)}_1 & \cdots & p^{(K)}_1 \\ 1 & p^{(1)}_2 & p^{(2)}_2 & \cdots & p^{(K)}_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & p^{(1)}_n & p^{(2)}_n & \cdots & p^{(K)}_n \end{bmatrix},
\]

\[
Y = \begin{bmatrix} d^{(1)}_1 \\ d^{(2)}_1 \\ \vdots \\ d^{(K)}_1 \end{bmatrix}^T,
\]

\[
\beta = \begin{bmatrix} \beta_{0j} \\ \beta_{1j} \\ \beta_{2j} \\ \cdots \\ \beta_{nj} \end{bmatrix}^T
\] (32)

An iterative divide-and-merge based least squares estimator [54], designed for solving large problems efficiently, is adopted for finding an optimal \( \beta \) in Eq. (31).

4. Parameter refinement

Now we have a rule base consisting of \( J \) fuzzy rules. The antecedent and consequent parameters associated with the rules were initialized, as described in the previous section. However, these parameters can be further refined through learning to improve the precision of these rules [50].

4.1. Type-2 fuzzy inference

Based on the \( J \) rules, a four-layer fuzzy neural network can be constructed as shown in Fig. 2. For a set of input values, \( [p_1, p_2, \ldots, p_s] \), a crisp system output \( \hat{y} \) can be obtained through the operation of the fuzzy neural network as follows:

- **Layer 1** contains \( J \) groups. Group \( j \) corresponds to rule \( R_j \) and contains 0 nodes. Node \( i \) of group \( j \) produces its output, \( \sigma^{(1)}_{ij} \), by computing the value of the corresponding membership function, i.e.,

\[
\sigma^{(1)}_{ij} = \mu_{\tilde{A}_{ij}}(p_i)
\] (33)

for \( 1 \leq i \leq n \) and \( 1 \leq j \leq J \). Note that each \( \sigma^{(1)}_{ij} \) is a type-1 fuzzy set.

- **Layer 2** contains \( J \) nodes. Node \( j \) produces its output, \( \sigma^{(2)}_j \), by computing the firing strength of rule \( R_j \), i.e.,

\[
\sigma^{(2)}_j = \sigma^{(1)}_{1j} \times \sigma^{(1)}_{2j} \times \cdots \times \sigma^{(1)}_{nj}
\] (34)
for $1 \leq j \leq J$, where the arithmetic product "\times" [55] is adopted for "AND". Obviously, each $\sigma_j^{(2)}$ is a type-1 fuzzy set.

- Layer 3 contains only one node. In this layer, the consequent value $y_j$ of each rule $R_j$ is computed as

$$y_j = \hat{\beta}_{0,j} + \hat{\beta}_{1,j}p_1 + \cdots + \hat{\beta}_{n,j}p_n$$

for $1 \leq j \leq J$. Next, $y_1, y_2, \ldots, y_J$ are sorted in ascending order, and let the resulting sequence be $y_{j1}, y_{j2}, \ldots, y_{jJ}$. Obviously, the following set

$$A = \sum_{i=1}^{J} y_{ji}$$

is a type-2 fuzzy set defined on the discrete universal set $\{y_{j1}, y_{j2}, \ldots, y_{jJ}\}$. Note that $\hat{\alpha}_j^{(2)}, i = 1, 2, \ldots, J$, are the inputs to the node of this layer, weighted by $(y_{j1}, y_{j2}, \ldots, y_{jJ})$, respectively. The node computes the centroid of Eq. (36) and its output $o^{(3)}$ is the result of Eq. (7).

- Layer 4 contains only one node which performs defuzzification on $o^{(3)}$ and produces a crisp output $o^{(4)}$ according to Eq. (14).

The crisp system output $\hat{y}$ corresponding to the input $[p_1, p_2, \ldots, p_n]$ is then set to be $o^{(4)}$.

### 4.2. Hybrid learning

We have $J$ rules. Each rule has 3n antecedent parameters. So we have 3J antecedent parameters in total, i.e., $m_{i,1}, \sigma_{i,1}^p, \sigma_{i,1}^s$, $m_{i,1}^p, \sigma_{i,1}^p, \sigma_{i,1}^s$, $m_{i,1}^p, \sigma_{i,1}^p, \sigma_{i,1}^s$, $\ldots$, $m_{n,1}^p, \sigma_{n,1}^p, \sigma_{n,1}^s$, $m_{i,1}^p, \sigma_{i,1}^p, \sigma_{i,1}^s$, $m_{i,J}^p, \sigma_{i,J}^p, \sigma_{i,J}^s$, $m_{i,J}^p, \sigma_{i,J}^p, \sigma_{i,J}^s$, $\ldots$, $m_{n,J}^p, \sigma_{n,J}^p, \sigma_{n,J}^s$. Also, each rule has $n+1$ consequent parameters. So we have $(n+1)J$ consequent parameters in total, i.e., $\hat{\beta}_{0,1}, \hat{\beta}_{1,1}, \ldots, \hat{\beta}_{n,1}, \ldots, \hat{\beta}_{0,J}, \hat{\beta}_{1,J}, \ldots, \hat{\beta}_{n,J}$. We adopt particle swarm optimization (PSO) to optimize the antecedent parameters and least-squares estimation (LSE) to optimize the consequent parameters. In each iteration of learning, both PSO and LSE are applied. We first treat all the consequent parameters as fixed and use PSO to refine the antecedent parameters. Then we treat all the antecedent parameters as fixed and use LSE to refine the consequent parameters. The process is iterated until the desired approximation precision is achieved.

PSO is a population-based global search algorithm for problem solving, proposed by Kennedy and Eberhart in 1995 [56]. It starts with a swarm of $S$ particles. Let the position and velocity of particle $i$ in iteration $t$ be denoted by $p_i^{(t)}$ and $v_i^{(t)}$, respectively. In iteration $t+1$, the velocity of particle $i, 1 \leq i \leq S$, is updated by

$$v_i^{(t+1)} = w \times v_i^{(t)} + k_1 \times (p_{\text{best}}^{(t)} - p_i^{(t)}) + k_2 \times rand_2 \times (\text{Gbest} - p_i^{(t)})$$

(37)

where $w$, $k_1$, and $k_2$ are constants, $rand_1$ and $rand_2$ are uniformly distributed random numbers in $[0, 1]$, $p_{\text{best}}^{(t)}$ is the best previous position of particle $i$, and $\text{Gbest}$ is the overall best particle position so far. Then the position of particle $i$ is updated by

$$p_i^{(t+1)} = p_i^{(t)} + v_i^{(t+1)}$$

(38)

for $1 \leq i \leq S$.

To refine consequent parameters, we treat all antecedent parameters as fixed. For training pattern $r$, i.e., $(p_i^{(t)}, d_i^{(t)}), 1 \leq r \leq \ell$, let $o_{i,1}^{(1)}(r)$, $o_{i,j}^{(2)}(r)$, and $o_{i,j}^{(4)}(r)$ denote the actual output of layers 1, 2, and 4, respectively, for this pattern. We have

$$o_{i,1}^{(1)}(r) = \mu_{i,j}^s [p_i^{(t)}] = \text{gauss}(u; \text{gauss}(p_i^{(t)}; m_{i,j}^p, \sigma_{i,j}^p), \sigma_{i,j}^s),$$

(39)

$$o_{i,j}^{(2)}(r) = o_{i,j}^{(1)}(r) \times o_{i,j}^{(1)}(r) \times \cdots \times o_{i,J}^{(1)}(r),$$

(40)
By Eq. (14), we have
\[
\alpha^{(4)}(r) = \frac{\sum_{s=1}^{n} \alpha_s (L^{(r)} + \rho_s^{(r)})}{2 \sum_{s=1}^{n} \alpha_s}.
\]
(41)

Also, we have \( y_j^{(r)} = \sum_{t=0}^{n} \beta_{t,j} p_t^{(r)} \) being the consequent part of rule \( R_j \) for \( p^{(r)} \). By a laborious derivation, we have the following linear system:
\[
X\beta = Y \tag{42}
\]

where
\[
X = \begin{bmatrix}
    x_{0,1}^{(1)} & x_{1,1}^{(1)} & \cdots & x_{n,1}^{(1)} \\
    x_{0,2}^{(2)} & x_{1,2}^{(2)} & \cdots & x_{n,2}^{(2)} \\
    \vdots & \vdots & \ddots & \vdots \\
    x_{0,J}^{(J)} & x_{1,J}^{(J)} & \cdots & x_{n,J}^{(J)} \\
\end{bmatrix},
\]
\[
\beta = \begin{bmatrix}
    \beta_{0,1} \\
    \beta_{1,1} \\
    \vdots \\
    \beta_{n,J} \\
\end{bmatrix}^T,
\]
\[
Y = \begin{bmatrix}
    d^{(1)} \\
    d^{(2)} \\
    \vdots \\
    d^{(L)} \\
\end{bmatrix}^T.
\]

with the sizes \( \ell \times (n+1)J \), \( (n+1)J \times 1 \), and \( \ell \times 1 \), respectively. Note that the entries in matrix \( X \) are related to \( \alpha \)-values, training patterns, and type-2 fuzzy sets and \( y_j^{(r)} \) in the rules. Again, the least squares estimator used in Section 3.2 is adopted to find an optimal \( \beta \) to Eq. (42).

Note that a neural network in the form of Fig. 2 created for the \( J \) rules is used for hybrid learning. The antecedent and consequent parameters of the \( J \) rules are refined through the operation of the neural network. In PSO, when a new position is derived, the corresponding RMSE should be calculated using the neural network. Based on the results, the Gbest and Pbest in Eq. (37) are updated. In LSE, the entries of the matrix \( X \) in Eq. (42) are obtained using the neural network. In PSO, the consequent parameters are kept fixed, and we only update antecedent parameters. In LSE, the antecedent parameters are kept fixed, and we only update consequent parameters.

4.3. Time complexity

A brief complexity analysis is given here [50]. Let the population size involved in PSO be \( P \), and the number of iterations required by hybrid learning is \( I \). For an input pattern, it requires \( (4nJ + 8DNJ) \) arithmetic operations, e.g., additions and multiplications, to compute the outputs of Layer 1. To get the outputs of Layer 2, 2DNJ operations are required. It requires about \( 6J(4D + 2I) \) operations to compute the output of Layer 3. Finally, about 3D operations are required for defuzzification in Layer 4. Therefore, it requires in total
\[
\ell(2nJ + 10DNJ + 24D) + 12J + 3D
\]
operations to compute the crisp outputs of all \( \ell \) training patterns. Furthermore, it requires \( \ell LnJ \) operations to find an optimal solution to Eq. (42), where \( L \) is a constant. Therefore, a particle in PSO requires
\[
\ell(2nJ + 10DNJ + 24D) + 12J + 3D + L\ell nJ
\]
operations involved in one iteration. For \( P \) particles and \( I \) iterations, the total time complexity involved in hybrid learning is
\[
P_t[I(\ell(2nJ + 10DNJ + 24D) + 12J + 3D) + L\ell nJ]
\]
which is of order \( O(\ell DNJ + \ell J^2) \).

5. Example

We give an example here for illustration. Suppose we have a training dataset containing 10 patterns \( \{p^{(1)}, d^{(1)}, p^{(2)}, d^{(2)}, \ldots, p^{(10)}, d^{(10)}\} \) as follows:
\[
p^{(1)} = [0.25, 0.12], \quad d^{(1)} = 0.023; \\
p^{(2)} = [0.78, 0.83], \quad d^{(2)} = 0.310; \\
p^{(3)} = [0.32, 0.27], \quad d^{(3)} = 0.077; \\
p^{(4)} = [0.18, 0.19], \quad d^{(4)} = 0.018; \\
p^{(5)} = [0.64, 0.72], \quad d^{(5)} = 0.316; \\
p^{(6)} = [0.09, 0.21], \quad d^{(6)} = 0.005; \\
p^{(7)} = [0.82, 0.79], \quad d^{(7)} = 0.412; \\
p^{(8)} = [0.13, 0.24], \quad d^{(8)} = 0.012; \\
p^{(9)} = [0.85, 0.88], \quad d^{(9)} = 0.266; \\
p^{(10)} = [0.73, 0.86], \quad d^{(10)} = 0.227.
\]

Note that \( n = 2 \) and \( \ell = 10 \) for this example.

- Rule base construction. We run the self-constructing clustering algorithm on the training dataset with \( \rho = 0.8^2 = 0.64 \), \( r = 0.4 \), and \( \sigma_0 = [0.35, 0.35] \). When all the training patterns have been fed in, two clusters are obtained, and the rule base contains the following two initial fuzzy rules:

\[
R_1 : \quad \text{IF } x_1 \text { IS gauss}(u; \text{gauss}(x_1; 0.1940, 0.4420), 0.0442) \quad \text{AND } x_2 \text { IS gauss}(u; \text{gauss}(x_2; 0.2060, 0.4070), 0.0407) \quad \text{THEN } y \text { IS } -0.0711 + 0.2713 x_1 + 0.2207 x_2
\]

\[
R_2 : \quad \text{IF } x_1 \text { IS gauss}(u; \text{gauss}(x_1; 0.7640, 0.4330), 0.0433) \quad \text{AND } x_2 \text { IS gauss}(u; \text{gauss}(x_2; 0.8160, 0.4140), 0.0414) \quad \text{THEN } y \text { IS } 0.7778 + 0.9200 x_1 - 1.4393 x_2
\]

Note that \( u \in [0, 1] \). A fuzzy neural network can then be built, which can provide a crisp output for any given input. For example, consider \( p^{(1)} \) as the input. The inferred output for \( p^{(1)} \) from the above two rules is \( \hat{y}^{(1)} = \hat{y}^{(1)} = 0.0354 \). The square error induced is \( (d^{(1)} - \hat{y}^{(1)})^2 = 0.000015 \).

- Parameter refinement. We improve the precision of the two rules through the application of the hybrid learning algorithm. For PSO, the population size is set as 10, and the parameters \( w, k_1, \) and \( k_2 \) are set as 0.5, 1.0, and 1.0, respectively. The refined type-2 fuzzy rules obtained are as follows:

\[
R_1 : \quad \text{IF } x_1 \text { IS gauss}(u; \text{gauss}(x_1; 0.2487, 0.4854), 0.0259) \quad \text{AND } x_2 \text { IS gauss}(u; \text{gauss}(x_2; 0.2177, 0.7510), 0.3933) \quad \text{THEN } y \text { IS } -0.0700 + 0.2695 x_1 + 0.2168 x_2
\]

\[
R_2 : \quad \text{IF } x_1 \text { IS gauss}(u; \text{gauss}(x_1; 0.9749, 0.3084, 0.0348) \quad \text{AND } x_2 \text { IS gauss}(u; \text{gauss}(x_2; 1.0205, 0.3790), 0.0828) \quad \text{THEN } y \text { IS } 1.5140 + 0.9082 x_1 - 2.3080 x_2
\]

Again, consider \( p^{(1)} \) as the input. The inferred output for \( p^{(1)} \) from these two refined rules is \( \hat{y}^{(1)} = 0.0234 \). The square error induced is \( (d^{(1)} - \hat{y}^{(1)})^2 = 0.00000016 \) which is much smaller than the previous one obtained by the initial rules. Apparently, the refined rules give a better performance than the initial rules.

6. Experimental Results

In this section, we present several experiments to demonstrate the effectiveness of the type-2 neuro-fuzzy modeling approach in stock price prediction. The daily historical closing stock prices are collected from the Taiwan Stock Exchange Capitalization Weighted Stock Index (TAIEX) [33,57] as training and testing data. Other stock exchange indexes, such as the Dow Jones Indexes [31] and the NASDAQ Stock Exchange Indexes [30], are also used. In the training
phase, a rule base is constructed and refined. In the testing phase, the refined rules are used for stock price prediction.

We compare the performance of the type-2 neuro-fuzzy modeling approach with that of other approaches based on conventional regression [3,4], artificial neural networks [9,11,14–16], fuzzy time series [5–7], and support vector regression [24]. We adopt mostly a 10-month/2-month split for generating the training and testing data, which is the same experimental settings as in [5–7]. For instance, the daily stock closing prices from January 1999 to October 1999 are used for training and the daily stock closing prices from November 1999 to December 1999 are used for testing. For convenience, the type-2 neuro-fuzzy modeling approach is abbreviated as T2NFS in the following description. For the self-constructing clustering method, the values of \( \rho, \tau, \sigma_0, \) and \( \kappa \) are mostly set to be \( 0.1^\tau, 0.2, 0.1, \) and 0.1, respectively, where \( n \) is the number of input variables. For PSO, the population size and the parameters \( w, k_1, \) and \( k_2 \) are mostly set as 5, 0.5, 1.5, and 1.5, respectively. These parameters are specified through experiments. The values were chosen by running on a validation set which is small.

### 6.1. Experiment I

In this experiment, data are taken from TAIEX only. We follow [6] to prepare training and testing patterns to be used. Let \( \{s_1, s_2, \ldots, s_T\} \) be the daily stock closing prices of TAIEX. Pattern \( r, (\mathbf{p}^{(r)}, d^{(r)}), \) is defined by

\[
d^{(r)} = s_r,
\]

\[
\mathbf{p}^{(r)} = [s_1, \ldots, s_{r-1}].
\]

Note that \( \mathbf{p}^{(r)} \) is a one-dimensional vector. For each year in 1999–2004, we have one training dataset and one testing dataset. Each training dataset contains about 300 patterns and each testing dataset contains about 60 patterns.

Table 1 shows a performance comparison between T2NFS and some other methods. In this table, RMSE indicates “root mean square error” which is a frequently-used measure of the differences between the values predicted and the values actually observed. We list the measure for each year and for each method. Besides, we list the average RMSE for each method, showing the average of all the six years. Also, U_FTS stands for the univariate fuzzy time series model, U_R for the univariate conventional regression model, U NN for the univariate neural network model, U_NN_FTS for the univariate neural network-based fuzzy time series model, and U_NN_FTS_S for the univariate neural network-based fuzzy time series model with substitutes. The numbers listed in this table for these models are adopted from [6]. From this table, it is obvious that T2NFS achieves the best performance, having the least RMSE and average RMSE. For example, T2NFS gets 84.7 in average RMSE, while U_FTS gets 117.4, U_R gets 374.2, U_NN gets 145.0, U_NN_FTS gets 125.0, and U_NN_FTS_S gets 107.8.

A naive predictor which adopts “the closing price of tomorrow is the closing price of today” predicts the stock price with RMSE being 103, 130, 113, 66, 53, and 55 for year 1999, 2000, 2001, 2002, 2003, and 2004, respectively. Our T2NFS predicts with RMSE being 100, 128, 111, 64, 51, and 54 instead. Obviously, our T2NFS performs only slightly better than the naive predictor. In fact, the other methods in Table 1 perform much worse than the naive predictor. However, with more information as input, as shown in Experiment III and Experiment IV later, our T2NFS can perform much better than the naive predictor.

We investigate the effect of the settings of the parameters \( \rho, \tau, \sigma_0, \) and \( \kappa \) associated with the self-constructing clustering method on prediction performance of T2NFS. Table 2 shows the RMSE values obtained by T2NFS with \( \rho \) varied in the range between 0.1 and 0.5. Table 3 shows the RMSE values obtained by T2NFS with \( \tau \) varied in the range between 0.1 and 0.3. Table 4 shows the RMSE values obtained by T2NFS with \( \sigma_0 \) varied in the range between 0.05 and 0.25. Table 5 shows the RMSE values obtained by T2NFS with \( \kappa \) varied in the range between 0.05 and 0.25.

Table 2 Performance of T2NFS with different settings of \( \rho \) for Experiment I.

<table>
<thead>
<tr>
<th>( \rho )</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>100</td>
</tr>
<tr>
<td>0.2</td>
<td>101</td>
</tr>
<tr>
<td>0.3</td>
<td>101</td>
</tr>
<tr>
<td>0.4</td>
<td>100</td>
</tr>
<tr>
<td>0.5</td>
<td>99</td>
</tr>
</tbody>
</table>

Table 3 Performance of T2NFS with different settings of \( \tau \) for Experiment I.

<table>
<thead>
<tr>
<th>( \tau )</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>100</td>
</tr>
<tr>
<td>0.15</td>
<td>100</td>
</tr>
<tr>
<td>0.20</td>
<td>101</td>
</tr>
<tr>
<td>0.25</td>
<td>102</td>
</tr>
<tr>
<td>0.30</td>
<td>101</td>
</tr>
</tbody>
</table>

Table 4 Performance of T2NFS with different settings of \( \sigma_0 \) for Experiment I.

<table>
<thead>
<tr>
<th>( \sigma_0 )</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>101</td>
</tr>
<tr>
<td>0.10</td>
<td>100</td>
</tr>
<tr>
<td>0.15</td>
<td>102</td>
</tr>
<tr>
<td>0.20</td>
<td>101</td>
</tr>
<tr>
<td>0.25</td>
<td>101</td>
</tr>
</tbody>
</table>

Table 5 Performance of T2NFS with different settings of \( \kappa \) for Experiment I.

<table>
<thead>
<tr>
<th>( \kappa )</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>101</td>
</tr>
<tr>
<td>0.10</td>
<td>100</td>
</tr>
<tr>
<td>0.15</td>
<td>101</td>
</tr>
<tr>
<td>0.20</td>
<td>101</td>
</tr>
<tr>
<td>0.25</td>
<td>102</td>
</tr>
</tbody>
</table>
of 12 months, about 360 test patterns are involved, which are more than training patterns. For some years, the RMSE is better, while for the others, the RMSE is worse.

6.2. Experiment II

In this experiment, we follow [6] to prepare training and testing patterns to be used. In addition to TAIEX, the Taiwan Futures Exchange (TAIFEX) [58] index is used as extra information. Let \( \{ s_1, s_2, \ldots, s_r \} \) and \( \{ w_1, w_2, \ldots, w_1 \} \) be the daily stock closing prices of TAIEX and TAIFEX, respectively. Pattern \( r, (p^{(r)}, d^{(r)}) \), is defined by

\[
d^{(r)} = s_r, \quad (46)
\]

\[
p^{(r)} = [s_{r-1}, w_{r-1}]. \quad (47)
\]

Note that \( p^{(r)} \) is a two-dimensional vector.

Table 7 shows a performance comparison between T2NFS and some other methods. In this table, B.R stands for the bivariate conventional regression model, B.NN for the bivariate neural network model, B.NN_FTS for the bivariate neural network-based fuzzy time series model, and B.NN_FTS_S for the bivariate neural network-based fuzzy time series model with substitutes. The numbers listed in this table for these models are adopted from [6]. From this table, we can see that T2NFS achieves the best performance in RMSE and average RMSE. Also, from Tables 1 and 7, we can see that the extra information taken from TAIFEX can slightly improve forecasting precision.

We investigate the effect of the settings of the parameters population size (\( P \)), \( w \), \( k_1 \), and \( k_2 \) associated with PSO on prediction performance of T2NFS. Table 8 shows the RMSE values obtained by T2NFS with \( P \), varied in the range between 5 and 25. Table 9 shows the RMSE values obtained by T2NFS with \( w \) varied in the range between 0.1 and 0.9. Table 10 shows the RMSE values obtained by T2NFS with \( k_1 \) and \( k_2 \) varied in the range between 0.1 and 2.0.

6.3. Experiment III

In this experiment, we follow [5,7] to prepare training and testing patterns to be used. In addition to TAIEX, the Dow Jones Industrial Average (DJIA) [31] and the NASDAQ Composite Index (NASDAQ) [30] are used as extra information. Let \( \{ s_1, s_2, \ldots, s_r \} \), \( \{ w_1, w_2, \ldots, w_1 \} \), and \( \{ q_1, q_2, \ldots, q_r \} \) be the daily stock closing prices of TAIEX, DJIA, and NASDAQ, respectively. Pattern \( r, (p^{(r)}, d^{(r)}) \), is defined in three ways. In all these three ways, \( d^{(r)} \) is given by

\[
d^{(r)} = s_r. \quad (48)
\]

However, \( p^{(r)} \) is defined by

- Case 1: \( p^{(r)} = [s_{r-1}, w_{r-1}, q_{r-1}] \) which combines TAIEX and DJIA,
- Case 2: \( p^{(r)} = [s_{r-1}, q_{r-1}] \) which combines TAIEX and NASDAQ, or
- Case 3: \( p^{(r)} = [s_{r-1}, w_{r-1}, q_{r-1}] \) which combines TAIEX, DJIA, and NASDAQ.

Note that the first two are two-dimensional, while the last is three-dimensional.

Table 11 shows a performance comparison between T2NFS and some other methods. In this table, FTS_{A^*, B} stands for the forecasting model based on fuzzy time series [5] and FTS_{FVG, A*} for the one based on fuzzy time series and fuzzy variation groups [7]. The ending with \( D, N \), or \( DN \) indicates that \( p^{(r)} \) is in the form of case 1, case 2, or case 3, respectively. For instance, FTS_{D, N} indicates that the fuzzy time series model is used and \( p^{(r)} \) is in the form of case 1. From this table, it is obvious that T2NFS achieves the best performance in RMSE and average RMSE. Note that the methods of T2NFS only provide marginal improvements over other methods in the year 2003 data set. The reason could be that the variation of the prices in the testing period of year 2003 is comparatively small. The difference between the maximum price and the minimum price is 400, the smallest among all the testing years. Therefore, prediction is easier and all methods are more likely to have a good prediction for this year. Fig. 3 shows the predicted results by T2NFS_D, in the six testing periods for TAIEX.

6.4. Experiment IV

In this experiment, we compare our system with SVM-based ones. We follow [24] to prepare training and testing patterns to be used. Let \( \{ s_1, s_2, \ldots, s_r \} \) be the daily stock closing prices of TAIEX.

Table 8

<table>
<thead>
<tr>
<th>( P )</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>2000</td>
</tr>
<tr>
<td>5</td>
<td>93</td>
</tr>
<tr>
<td>10</td>
<td>93</td>
</tr>
<tr>
<td>15</td>
<td>102</td>
</tr>
<tr>
<td>20</td>
<td>98</td>
</tr>
<tr>
<td>25</td>
<td>96</td>
</tr>
</tbody>
</table>

Table 9

<table>
<thead>
<tr>
<th>( w )</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>2000</td>
</tr>
<tr>
<td>0.1</td>
<td>100</td>
</tr>
<tr>
<td>0.3</td>
<td>99</td>
</tr>
<tr>
<td>0.5</td>
<td>93</td>
</tr>
<tr>
<td>0.7</td>
<td>92</td>
</tr>
<tr>
<td>0.9</td>
<td>98</td>
</tr>
</tbody>
</table>

Table 10

<table>
<thead>
<tr>
<th>( k_1, k_2 )</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>2000</td>
</tr>
<tr>
<td>[0.1, 0.1]</td>
<td>97</td>
</tr>
<tr>
<td>[0.5, 0.5]</td>
<td>93</td>
</tr>
<tr>
<td>[1.0, 1.0]</td>
<td>99</td>
</tr>
<tr>
<td>[1.5, 1.5]</td>
<td>93</td>
</tr>
<tr>
<td>[2.0, 2.0]</td>
<td>99</td>
</tr>
</tbody>
</table>
Fig. 3. Predicted results by T2NFS_DN for Experiment III.
Two technical indices, SMA and BIAS, are defined. SMA, abbreviated for simple moving average, is used to emphasize the direction of a trend and to smooth out price and volume fluctuations. The \( m \)-day SMA of the \( t \)-th day is defined as follows:

\[
\text{SMA}_m(t) = \frac{\sum_{i=-m}^{t-m} S_i}{m}.
\]

BIAS is used to observe the difference between the closing price and the moving average line. The \( m \)-day BIAS of the \( t \)-th day is defined as follows:

\[
\text{BIAS}_m(t) = \frac{S_t - \text{SMA}_m(t)}{\text{SMA}_m(t)} \times 100.
\]

Pattern \( r \), \( (\mathbf{p}^r, \mathbf{d}^r) \), is then defined as

\[
\mathbf{d}^r = S_r,
\]

\[
\mathbf{p}^r = [\text{SMA}_{60}(r - 1), \text{BIAS}_{60}(r - 1)].
\]

Note that \( \mathbf{p}^r \) is a two-dimensional vector.

Table 12 shows a performance comparison between T2NFS and some other methods. In this table, SKSVar stands for support vector regression (SVR) with single kernel [24], and MKSVR for SVR with multiple kernels [24]. For SKSVar, three parameters, \( \gamma \), \( \varepsilon \), and \( C \), have to be determined in advance. We examine the performance of SKSVar with \( C = 1 \) and \( \varepsilon = 0.01 \). Besides, we try with 23 different settings of \( \gamma \): from 0.1 to 0.9 with a stepping factor of 0.1, from 1 to 9 with a stepping factor of 1, and from 10 to 50 with a stepping factor of 10. The best RMSE values obtained by SKSVar are listed in Table 12. For multiple-kernel learning, a kernel combining all the previous 23 different RBF kernels is considered. Therefore, the combined kernel matrix is a weighted sum of 23 kernel matrices. The RMSE values obtained by MKSVR for the six datasets are also listed in the table. MKSVar performs better than the best SKSVar for each year. MKSVar performs better than T2NFS for three years, 1999, 2003, and 2004. However, T2NFS achieves the best performance for the other three years, 2000, 2001, and 2002. Besides, T2NFS has the least average RMSE.

6.5. Experiment V

Finally, we compare T2NFS with some other methods on NASDAQ data using two more other error measures, mean absolute error (MAE) and mean absolute percentage error (MAPE) [44]. The data is taken from the period between October 7, 2008 and June 26, 2009. The data of the first 146 days is used for training, while the data of the last 36 days is used for testing. The results are shown in Table 13. In this table, MLP stands for multi-layer perceptrons [10], DAN2 for dynamic architecture for artificial neural networks [12]. We use the previous four days to forecast the fifth day for MLP, DAN2, and T2NFS. GARCH-MLP and GARCH-DAN2 are MLP and DAN2, respectively, with two additional inputs calculated from GARCH [4]. As shown in the table, T2NFS performs best in terms of each different error measure.

7. Conclusion

We have presented an application of type-2 neuro-fuzzy modeling to stock price prediction. Type-2 fuzzy rules are generated automatically by a self-constructing clustering method and the obtained type-2 fuzzy rules are refined by a hybrid learning algorithm. The self-constructing clustering method partitions the training data set into clusters. The membership function associated with each cluster is defined with the mean and deviation of the data points included in the cluster. A type-2 TSK rule is derived from each cluster to form a fuzzy rule base. Then the hybrid learning algorithm, which incorporates particle swarm optimization and least squares estimation, is used to refine the antecedent parameters and the consequent parameters, respectively, associated with the rules.

We adopted the closing prices of the previous day as input in most of the experiments in Section 6. This seemingly simple adoption may not result in good results, although our proposed type-2 neuro-fuzzy modeling approach works favorably well compared with other methods. A much more refined feature selection step is required to provide relevant historic closing prices and other indicators as input for a prediction system to get better stock price prediction [59].

Acknowledgements

This work was partially supported by “Aim for the Top University Plan” of the National Sun Yat-Sen University and Ministry of Education, and by the National Science Council under the grants NSC-98-2221-E-110-052 and NSC-99-2622-E-110-007-CC3. The authors are grateful to the anonymous reviewers for their comments which were very helpful in improving the quality and presentation of the paper.