A Survey of Forecasting Preprocessing Techniques using RNs

J.M. Górriz and J.C. Segura-Luna
Dpt. Signal Theory Daniel Saucedo s/n E-18071
gorriz@ugr.es web: hal.ugr.es

C.G. Puntonet and M. Salmerón
Dpt. Architecture and Computer Tech. Daniel Saucedo E-18071
carlos@atc.ugr.es web: atc.ugr.es

Keywords: Regularization Networks, Independent Component Analysis, Principal Component Analysis

Received: October 28, 2004

In this paper we make a survey of various preprocessing techniques including the statistical method for volatile time series forecasting using Regularization Networks (RNs). These methods improve the performance of Regularization Networks i.e. using Independent Component Analysis (ICA) algorithms and filtering as preprocessing tools. The preprocessed data is introduced into a Regularized Artificial Neural Network (ANN) based on radial basis functions (RBFs) and the prediction results are compared with the ones we get without these preprocessing tools, with the high computational effort method based on multidimensional regularization networks (MRN) and with the Principal Component Analysis (PCA) technique.

Povzetek:

1 Introduction

In the history of research of the forecasting problem one can extract various relevant periods such as the following mentioned: a possible solution to this problem was described by Box and Jenkins [1], who developed a time-series forecasting analysis technique based on linear systems. Basically the procedure consisted of suppressing the non-seasonality of the series, performing parameter analysis, which measures time-series correlation, and selecting the model that best fits the data set (a specific order ARIMA model). But in real systems, non-linear and stochastic phenomena crop up, and then time series dynamics cannot be described exactly using classical models. ANNs have improved results in forecasting by detecting the non-linear nature of the data. ANNs based on RBFs allow a better forecasting adjustment; they implement local approximations to non-linear functions, minimizing the mean square error to achieve the adjustment of neural parameters. For example, Platt's algorithm [2], Resource Allocating Network (RAN), consisted of neural network size control, reducing the computational time cost associated with computing the optimum weights in perceptron networks.

Matrix decomposition techniques have been used as an improvement on Platt's model [3]. For example, Singular Value Decomposition (SVD) with pivoting QR decomposition selects the most relevant data in the input space avoiding non-relevant information processing (NAPA-PRED "Neural model with Automatic Parameter Adjustment for PREDiction"). NAPA-PRED also includes neural pruning [4]. An improved version of this algorithm can be found in [5] based on Support Vector Machine philosophy.

The next step was to include exogenous information in these models. There are some choices in order to do that; we can use the forecasting model used in [6] which gives good results but with computational time and complexity cost; Principal Component Analysis (PCA) is a well-established tool in Finance. It was already proved [3] that prediction results can be improved using the PCA technique. This method linear transform the observed signal into principal components which are uncorrelated (features), giving projections of the data in the direction of the maximum variance [7]. PCA algorithms use only second order statistical information; Finally, in [8] we can discover interesting structure in finance using the new signal-processing tool Independent Component Analysis (ICA). ICA finds statistically independent components using higher order statistical information for blind source separation ([9], [10]). This new technique may use Entropy (Bell and Sejnowski 1995, [11]), Contrast functions based on Information Theory (Comon 1994, [12]), Mutual Information (Amari, Cichocki y Yang 1996, [13]) or geometric considerations in data distribution spaces (Carlos G. Puntonet 1994 [14], [15]), etc. Forecasting and analyzing financial time series using ICA can contributes to a better understanding and prediction of financial markets ([6],[8]).

There exist numerous forecasting applications in time series forecasting, as analyzed in [16]: signal statistical preprocessing and communications, industrial control processing, econometrics, meteorology, physics, biology, medicine, oceanography, seismology, astronomy and psychology. We organize the essay as follows. In section 2 we describe the neural model used and the certain conditions to achieve a good confidence interval in prediction. In sections 3,4 and 5 we describe in detail three methods for time series preprocessing showing some results and finally in section we describe a brand new experimental framework comparing the previous discussed methods stating some conclusions.

2 Regularization networks based on RBFs

Because of their inherent non-linear processing and learning capabilities, ANNs (Artificial Neural Networks) have been proposed to solve prediction problems. An excellent survey of NN forecasting applications is to be found in [17]. There it is claimed that neural nets often offer better performance, especially for difficult time series than are hard to deal with classical models such as ARIMA models [1]. One of the simplest, but also most powerful, ANN models is the Radial Basis Function (RBF) network model. This consists of locally-receptive activation functions (or neurons) implemented by means of gaussian functions [18]. In mathematical terms, we have

$$o(\mathbf{x}) = \sum_{i=1}^{N} o_i(\mathbf{x}) = \sum_{i=1}^{N} h_i \exp\left\{-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{\sigma_i^2}\right\} \quad (1)$$

where N denotes the number of nodes (RBFs) used; $o_i(\mathbf{x})$ gives the output computed by the i-th RBF for the input vector $\mathbf{x}$ as an exponential transformation over the norm that measures the distance between $\mathbf{x}$ and the RBF center $\mathbf{c}_i$, whereas σ_i denotes the *radius* that controls the locality degree of the corresponding i-th gaussian response. The global output $o(\mathbf{x})$ of the neural network is, as can be seen, a linear aggregate or combination of the individual outputs, weighted by the real coefficients h_i.

In most RBF network applications, the coefficients h_i are determined after setting up the location and radius for each of the N nodes. The locations can be set, as in [18], by a *clustering* algorithm, such as the *K-means algorithm* [19], and the radius is usually set after taking into account considerations on RBFs close to the one being configured. The adjustment of the linear expansion coefficients can be done using recursive methods for linear least squares problems [20, 21] or the new method based on Regularization-VC Theory presented in [5] characterized by a suitable regularization term which enforces flatness in the input space, so that the actual risk functional over a training data set is minimized, and determined by the previously set parameter values [22]. Recursive specification allows for real-time implementations, but the questions arises of whether or not we are using a simplified-enough neural network, and this is a question we will try to address using matrix techniques over the data processed by the neural net.

In the particular context of the RBF networks, the mapping of a time series prediction problem to the network is performed setting up the input as past values (consecutive ones, in a first approximation) of the time series. The output is viewed as a prediction for the future value that we want to estimate, and the computed error between the desired and network- estimated value is used to adjust parameters in the network.

2.1 Regularization Theory (RT)

RT appeared in the methods for solving *ill posed problems* [23]. In RT we minimize a expression similar to the one in Support Vector Machines scenario (SVM). However, the search criterium is enforcing smoothness (instead of flatness) for the function in input space (instead of feature space). Thus we get:

$$R_{reg}[f] = R_{emp}[f] + \frac{\lambda}{2}||\hat{P}f||^2. \quad (2)$$

where $\hat{P}$ denotes a regularization operator in the sense of [23], mapping from the Hilbert Space H of functions to a dot product Space D such as $\langle f, g\rangle \quad \forall f, g \in H$ is well defined. Applying Fréchett's differential[1] to equation 2 and the concept of Greent's function of $\hat{P}^*\hat{P}$:

$$\hat{P}^*\hat{P} \cdot G(x_i, x_j) = \delta(x_i - x_j). \quad (3)$$

(here δ denotes the Diract's δ, that is $\langle f, \delta(x_i)\rangle = f(x_i)$), we get [22]:

$$f(x) = \lambda \sum_{i=1}^{\ell} [y_i - f(x_i)]_\epsilon \cdot G(x, x_i). \quad (4)$$

The correspondence between SVM and RN is proved if and only if the Greent's function G is an "admissible" kernel in the terms of Mercert's theorem [24],i.e. we can write G as:

$$G(x_i, x_j) = \langle\Phi(x_i), \Phi(x_j)\rangle \quad (5)$$

$$with \quad \Phi : x_i \rightarrow (\hat{P}G)(x_i, .). \quad (6)$$

Prove: Minimizing $||\mathbf{P}f||^2$ can be expressed as:

$$||\mathbf{P}f||^2 = \int dx(\mathbf{P}f)^2 = \int dx f(x)\mathbf{P}^*\mathbf{P}f(x) \quad (7)$$

we can expand f in terms of green's function associated to $\mathbf{P}$, thus we get:

$$\begin{aligned} ||\mathbf{P}f||^2 &= \textstyle\sum_{i,j}^N h_i h_j \int dx G(x, x_i)\mathbf{P}^*\mathbf{P}G(x, x_j) \\ &= \textstyle\sum_{i,j}^N h_i h_j \int dx G(x, x_i)\delta(x - x_j) \\ &= \textstyle\sum_{i,j}^N h_i h_j G(x_j, x_i) \end{aligned} \quad (8)$$

then only if G is a Mercer Kernel it correspond to a dot product in some feature space. Then minimizing 2 is equivalent to SVM minimization†.

A similar prove of this connection can be found in [25]. Hence given a regularization operator, we can find an admissible kernel such that SV machine using it will enforce flatness in feature space and minimize the equation 2. Moreover, given a SV kernel we can find a regularization operator such that the SVM can be seen as a RN.

[1] Generalized differentiation of a function: $dR[f] = \left[\frac{d}{d\rho}R[f + \rho h]\right]$, where $h \in H$.

2.2 On-line Endogenous Learning Machine Using Regularization Operators

In this section we show on-line RN based on "Resource Allocating Network" algorithms (RAN) [2] [2] which consist of a network using RBFs, a strategy for allocating new units (RBFs), using two part novelty condition [2]; input space selection and neural pruning using matrix decompositions such as SVD and QR with pivoting [4]; and a learning rule based on SRM as discussed in the previous sections. The pseudo-code of the new on-line algorithm is presented in [26]. Our network has 1 layer as is stated in equation 1. In terms of RBFs the latter equation can be expressed as:

$$f(x) = \sum_{i=1}^{N(t)} h_i \cdot \exp\left(-\frac{||x(t) - x_i(t)||^2}{2\sigma_i^2(t)}\right) + b. \tag{9}$$

where $N(t)$ is the number of neurons, $x_i(t)$ is the center of neurons and $\sigma_i(t)$ the radius of neurons, at time "t".

In order to minimize equation 2 we propose a regularization operator based on SVM philosophy. We enforce flatness in feature space, as described in [26], using the regularization operator $||\hat{P}f||^2 \equiv ||\omega||^2$, thus we get:

$$R_{reg}[f] = R_{emp}[f] + \frac{\lambda}{2} \sum_{i,j=1}^{N(t)} h_i h_j k(x_i, x_j). \tag{10}$$

We assume that $R_{emp} = (y - f(x))^2$ we minimize equation 10 adjusting the centers and radius (gradient descend method $\Delta\chi = -\eta \frac{\partial R[f]}{\partial \chi}$, with simulated annealing [27]):

$$\begin{array}{l} \Delta x_i = -2\frac{\eta}{\sigma_i}(x - x_i)h_i(f(x) - y)k(x, x_i) \\ \quad +\alpha \sum_{i,j=1}^{N(t)} h_i h_j k(x_i, x_j)(x_i - x_j). \end{array} \tag{11}$$

and

$$\Delta h_i = \tilde{\alpha}(t) f(x_i) - \eta(f(x) - y)k(x, x_i). \tag{12}$$

where $\alpha(t), \tilde{\alpha}(t)$ are scalar-valued "adaptation gain", related to a similar gain used in the stochastic approximation processes, as in these methods, it should decrease in time. The second summand in equation 11 can be evaluated in several regions inspired by the so called "divide-and-conquer" principle and used in unsupervised learning, i.e. competitive learning in self organizing maps [28] or in SVMs experts [29]. This is necessary because of volatile nature of time series, i.e. stock returns, switch their dynamics among different regions, leading to gradual changes in the dependency between the input and output variables [26]. Thus the super-index in the latter equation is redefined as:

$$N_c(t) = \{s_i(t) : ||x(t) - x_i(t)|| \leq \rho\}. \tag{13}$$

that is the set of neurons close to the current input.

3 RNs and PCA

3.1 Introduction

PCA is probably the oldest and most popular technique in multivariate data analysis. It transforms the data space into a feature space, in such a way, that the new data space is represented by a reduced number of "effective" features. Its main advantages lie in the low computational effort and the algebraic procedure.

Given a $n \times N$ data set $\mathbf{x}$,where N is the sample size, PCA tries to find a linear transformation $\tilde{\mathbf{x}} = \mathbf{W}^T\mathbf{x}$ into a new orthogonal basis $\mathbf{W} = \{\mathbf{w}_1, \ldots, \mathbf{w}_m\} \quad m \leq n$ such that:

$$Cov(\tilde{\mathbf{x}}) = E\{\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\} = \mathbf{W}^T Cov(\mathbf{x})\mathbf{W} = \mathbf{\Lambda} \tag{14}$$

where $\mathbf{\Lambda} = diag(\lambda_1, \ldots, \lambda_n)$ is a diagonal matrix. Hence PCA, decorrelates the vector $\mathbf{x}$ as all off-diagonal elements in the covariance matrix of the transformed vector vanish. In addition to the transformation presented in equation 14 (Karhunen-Loeve transformation when $m = n$) the variances of the transformed vectors $\tilde{\mathbf{x}}$ can be normalized to one using:

$$\tilde{\mathbf{x}} = \mathbf{W}_z^T\mathbf{x} \tag{15}$$

with the sphering matrix $\mathbf{W}_z = \left[\frac{\mathbf{w_1}}{\sqrt{\lambda_1}}, \ldots, \frac{\mathbf{w_m}}{\sqrt{\lambda_m}}\right]$ It has been shown that prediction results can be improved using this technique in [4].

In this Section we give an overview of the basic ideas underlying Principal Component Analysis (PCA) and its application to improve forecasting results using the algorithm presented in Section 2. The improvement consist on including exogenous information as is shown [3] and extracting results from this technique to complete the different methods of inclusion extra information.

The purpose of this Section is twofold. It should serve as a self-contained introduction to PCA and its relation with ANNs (Section 3.2). On the other hand, in Section 3.3, we discuss the use of this tool with the algorithm presented in Section 2 to get better results in prediction. To this end we follow the method proposed in [3] and see the disadvantages of using it.

3.2 Basis PCA and Applications

There are numerous forecasting applications in which interesting relations between variables are studied. The nature of this dependence among observations of a time series is of considerable practical interest and researchers have to analyze this dependence to find out which variables are most relevant in practical problems. Thus, our objective is to obtain a forecast function of a time series from current and past values of relevant exogenous variables.

Some tools have been developed in physics and engineering areas which allow this kind of analysis. For example, Factor Analysis (FA) [30] is useful to extract relevant combinations (i.e factors) from the set of original variables.

[2] The principal feature of these algorithms is sequential adaptation of neural resources.

The extracted factors, obtained from an input linear model, are rotated to find out interesting structures in data. The procedure is based on correlation matrix between variables [3].

FA has strong restrictions on the nature of data (linear models), thus PCA is more useful to our application due to the low computational effort and the algebraic procedure as it is shown in the next Section. Reducing input space dimension (feature space) using PCA is of vital importance when working with large data set or with "on line" applications (i.e time series forecasting). The key idea in PCA, as we say latter, is *transforming* the set of correlated input space variables into a lower dimension set of new uncorrelated *features*. This is an advantage in physics and engineering fields where theret's a high computational speed demand in on-line systems (such as sequential time series forecasting).

In addition to these traditional applications in physics and engineering, this technique has been applied to economy, psychology, and social sciences in general. However, owing to different reasons [31], PCA has not been established in these fields as good as the others. In some fields PCA became popular , i.e. statistics or data mining using intelligent computational techniques [32]. Obviously, the new research in neural networks and statistical learning theory will bring applications in which PCA will be applied to reduce dimensionality or real-time series analysis.

3.2.1 PCA Operation

Let $\mathbf{x} \in \mathcal{R}^n$ representing a stochastic process. The target in PCA [33] is to find a unitary vector basis (norm equal to 1)

$$\{\mathbf{u}_j : j = 1, 2, \ldots, r\} \ , \tag{16}$$

where $r < n$, and with projections of this kind:

$$\mathbf{u}_j^T \cdot \mathbf{x} \tag{17}$$

have *maximum expected variance*, w.r.t all possible configurations (16). In other words, the first vector belonging to this basis, $\mathbf{u}_1$, must have the following property: $\mathbf{u}_1 \cdot \mathbf{x}$, considered as a random variable (since $\mathbf{x}$ is a random variable), has maximum variance between all possible linear combinations of the components of $\mathbf{x}$. At the same time, $\mathbf{u}_2$ is such that $\mathbf{u}_2 \cdot \mathbf{x}$ has maximum variance between all possible orthogonal directions to $\mathbf{u}_1$, and so on. The next unitary vectors are selected from the set of vector $\{\mathbf{w}\}$ satisfying:

$$\mathbf{w}^T \cdot \mathbf{u}_k = 0, k = 1, \ldots, j-1 \ \ and \ \ \mathbf{w}^T \cdot \mathbf{w} = 1 \ , \tag{18}$$

and then from this set $\{\mathbf{w}\}$, we choose them using

$$\mathbf{u}_j = \arg\max_{\mathbf{w}} E\left(Var[\mathbf{w}^T \cdot \mathbf{x}]\right) \ , \tag{19}$$

where $\mathbf{w}$ verifies (18).

Let a vector $\mathbf{x} \in \mathcal{R}^n$, the set of orthogonal projections with maximum variances $\mathbf{u}_j^T \cdot \mathbf{x}$ are given by:

$$\max_{\mathbf{u}_j} E\left(Var[\mathbf{u}_j^T \cdot \mathbf{x}]\right) = \lambda_j \ , \tag{20}$$

where λ_j is the j-th *eigenvalue* of the covariance matrix

$$\mathbf{R} \equiv E\left[(\mathbf{x} - \mu_{\mathbf{x}}) \cdot (\mathbf{x} - \mu_{\mathbf{x}})^T\right] \ , \tag{21}$$

that is a $n \times n$ square matrix. In the equation (21), $\mu_{\mathbf{x}}$ represents the stationary stochastic process mean which can be calculated using the set of samples $\mathbf{x}$. the eigenvalues λ_j can be calculated according the EIGD of matrix (21), which definition and properties are shown in [34].

Furthermore, it can be proved that the "principal components" from which we can get the maximum variances, are the eigenvectors of the covariance matrix $\mathbf{R}$. Note that $\mathbf{R}$ is semi-definite positive matrix thus all eigenvalues are positive real numbers including 0 $[0, +\infty)$ and their eigenvectors $\mathbf{u}_j$ satisfying:

$$\mathbf{R} \cdot \mathbf{u}_j = \lambda_j \cdot \mathbf{u}_j \ , \tag{22}$$

where λ_j denotes the associated eigenvalue, can be merged to compose an orthogonal matrix.

Hence we can estimate the covariance matrix $\mathbf{R}$ as:

$$\hat{\mathbf{R}} \equiv 1/(N-1) \cdot \mathbf{X} \cdot \mathbf{X}^T \ , \tag{23}$$

where

$$\mathbf{X} = [\mathbf{x}_1 - \bar{\mathbf{x}}, \mathbf{x}_2 - \bar{\mathbf{x}}, \ldots, \mathbf{x}_N - \bar{\mathbf{x}}] \tag{24}$$

is a $n \times N$ matrix including the set of N samples (or n-dimensional vectors) $\mathbf{x}_i$, with mean equal to $\bar{\mathbf{x}}$. Once the estimation of $\hat{\mathbf{R}}$ has been got, we can use EIGD, to obtain the following matrix:

$$\mathbf{U} = [\mathbf{u}_1 \mathbf{u}_2 \ldots \mathbf{u}_n] \tag{25}$$

including all eigenvectors in the columns, and the diagonal matrix Λ with the corresponding eigenvalues in the main diagonal.

Obviously if we choose the set of orthogonal and unitary vectors given by the equation (25), then theret's an unique correspondence between the input space matrix $\mathbf{X}$ in equation (24) and the $n \times N$ matrix $\mathbf{X}' = \mathbf{U}^T \cdot \mathbf{X}$. This transformation is invertible since $\mathbf{U}^{-1} = \mathbf{U}$, i.e $\mathbf{U}$ is orthogonal. All the process is based on a simple linear transformation into a new orthogonal basis such that the covariance matrix in the new system is diagonal.

[3] Factor analysis is a statistical approach that can be used to analyze interrelationships among a large number of variables and to explain these variables in terms of their common underlying dimensions ("factors"). The statistical approach involving finding a way of condensing the information contained in a number of original variables into a smaller set of dimensions (factors) with a minimum loss of information. It has been used in disciplines as diverse as chemistry, sociology, economics or psychology.

3.3 Time Series prediction with PCA

In this Section we show how Principal Component Analysis (PCA) technique can be hybridized with the algorithm presented in section 2 to improve prediction results, including exogenous information.

3.3.1 Data compression and reducing dimensionality

As we mentioned latter, PCA is a useful tool in the preprocessing step in data analysis, i.e. those techniques based on artificial neural networks considered in this work. In this way, there can be a PCA layer that compresses raw data from the sample set.

The basic idea is consider a large set of input variables and transforms it to a new set of variables containing without loss of much information [7]. This is possible due to that very often, multivariate data contains redundant information of 2^{nd} order.

On the other hand, the more dimension reduction we want to achieve the more fraction of original loss variance, in other words, we can lose much relevant information. Thus, under this conditions, the inclusion of exogenous information would contaminate prediction capacity of any system (neural or not). Hence, if PCA extract only the first r factors (in terms of variance) such that:

$$r = \min\{k : \sum_{i=1}^{k} \lambda_i \geq \rho \cdot tr(\hat{\mathbf{R}})\} \ , \tag{26}$$

only a fraction of ρ of the exogenous data overall variance will be kept. In the equation (26), $tr(\hat{\mathbf{R}})$ denotes *trace* of the estimated covariance matrix $\hat{\mathbf{R}}$ for the set of data (that is, adding its diagonal elements or individual variance components). Given that $\sum_{i=1}^{n} \lambda_i$ is the overall variance and PCA projection variances are given by the eigenvalues λ_i, if we consider the complete set of eigenvalues we would have the complete variance, so this way, if we select a subset of eigenvalues $r < n$, we would hold a fraction ρ (at least) of the overall variance of the exogenous data.

In this method of data compression using the maximum variance principle, PCA is basically regarded as a standard statistical technique. In neural networks research areas, the term *unsupervised Hebbian learning* [35, 36, 37] is usually used to refer this powerful tool in data analysis, such as *discriminant analysis* is used as a theoretical foundation to justify neural architectures (i.e. multilayer perceptrons or linear architectures) for classification [38].

The previous discussion explains the concept of *dimension reduction*, projecting onto r more relevant unitary vectors (that is, those ones that hold the bigger fraction of variance of data) is the way of develop this reduction since multiplying by $\mathbf{U}$ gives a $r \times N$ matrix. In addition, the column vectors in $\mathbf{U}$ can be seen as *feature vectors*, containing the principal characteristics of the data set; the projection onto $\mathbf{u}_j$ must be understood, in that case, such as a measure of certain characteristic in data samples. The transformation onto this new feature space is suitable way to analyse raw data, thus, in this Section, we will use this technique in the preprocessing step, before neural stages.

3.3.2 Improving neural input space

Moreover, PCA can be used to include exogenous information [3], in other words, we can increase *input space dimension* using variables related to the original series. The principal advantage of using PCA over straightforward inclusion is that PCA can reduce input space dimensionality without loss of much information (in terms of variance) included is such variables.

As we said latter, to reduce input space dimensionality using PCA, a fraction of information, i.e variance, must be rejected. In some cases, it can be a decision with unforeseeable consequences, thus, a conservative policy should be followed, i.e. using PCA variables such as "extra" variables to improve the prediction results. This is the key idea in this Section.

This rule based on "catalytic variables" is appropriate in a practical point of view. In fact in [7], hybridized with filtering techniques, is a success. In the following example, we show how this technique is applied to stock series obtaining noticeable improvements.

3.4 Results

In the following Section we show an example in which we applied hybrid models based on PCA and ANN originally discussed in [3]. We intend to forecast (with horizon equal to 1) stock series (indexes) of different Spanish banks and other companies during the same period.

3.4.1 Description and data set

We have specifically focussed on the IBEX35 index of Spanish stock, which we consider the most representative sample of Spanish stock movements. We have chosen seven relevant indexes such as *Banesto*, *Bankinter*, *BBVA*, *Pastor*, *Popular*, *SCH* y *Zaragozano*, and we build a matrix including 1672 consecutive observations (closing prices). A representation of theses indexes can be found in the figure 1.

Itt's clear, from the latter figure, that there is a wide range of indexes closing prices. Thus, it means the need for a logarithmic transformation (to make variance steady) and later suitable differentiation of the data (to remove the residual non-stationary behavior). Once the proper transformations are achieved we obtain the results shown in figure 2.

After these basic transformations, the new set of series can be processed using the algorithm introduced in Section 2. The object of the method is to train our neural network based on RBFs with the set of transformed series, to predict (with horizon equal to 1) the first of the selected stocks (strictly speaking, we predict the transformed value that must be inverted in the final step) using its own endogenous information (the number of lags were fixed to 2)

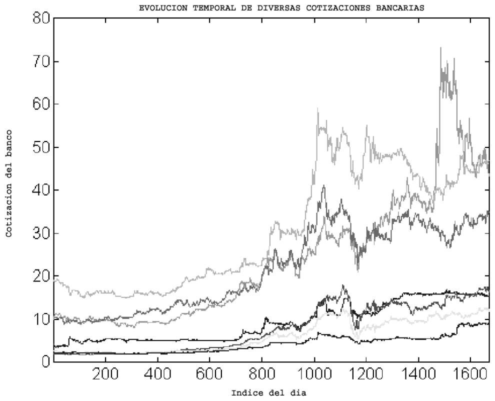

Figure 1: Closing prices evolution for selected indexes.

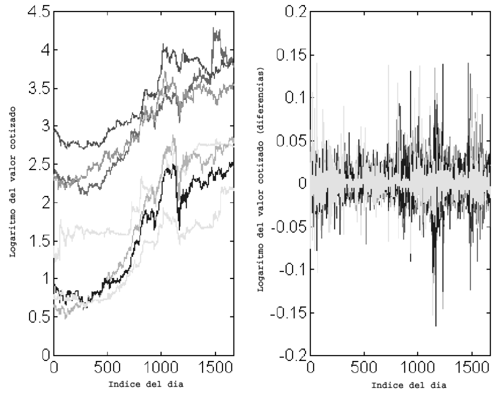

Figure 2: Closing prices evolution for selected indexes, after logarithmic transformation (on the left) and differentiation (on the right).

Table 1: Eigenvalues λ_i and variance percentages of PCs.

Index	*Eigenvalue*	*Pct. variance*	*Pct. overall*
1	0.9302	33.03	33.03
2	0.6670	23.69	56.72
3	0.3303	11.73	68.45
4	0.2694	9.57	78.02
5	0.2198	7.80	85.82
6	0.2134	7.58	93.40
7	0.1858	6.60	100.00

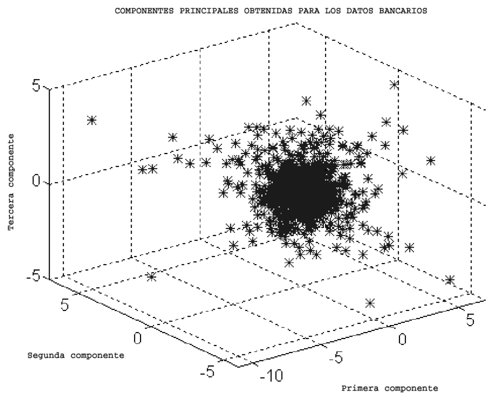

Figure 3: 3D schematic representation of the three first principal components.

and the more relevant principal components (thus we don't use the other series directly).

Using a training set consisting of 1000 samples, we computed the first 3 principal components. The matrix used consist of the complete set of series (every stock). As we mentioned in Section 3.3.2, we determinate the number of inputs to improve the prediction result of interest using the principal components as additional data input. We remark that the 3 components represents about 70% of the overall variance (table 1). In a three dimensional space we can plot the components as is shown in figure 3.

Finally, the last 10 samples of the complete set (1000) were used to compare prediction results with and without exogenous inputs. In addition, in this example we included the well-known sphering or y-score transformation [33](as the variance along all principal components equals one):

$$\mathbf{w}_i = \sqrt{\lambda_i}^{-1} \cdot \mathbf{u}_i \ , \tag{27}$$

instead of using the original eigenvector $\mathbf{u}_i$. This transformation is very common in the field of artificial neural networks and in many ICA-algorithms (whitening) in a preprocessing step as it completely removes all correlations up to the 2^{nd} order.

The reason for using just the last 10 sample points lies in the fact that economic series extremely volatile and PCA can only extract up to second order relations (in Section 4 we use a better tool to develop it). So the extra infor-

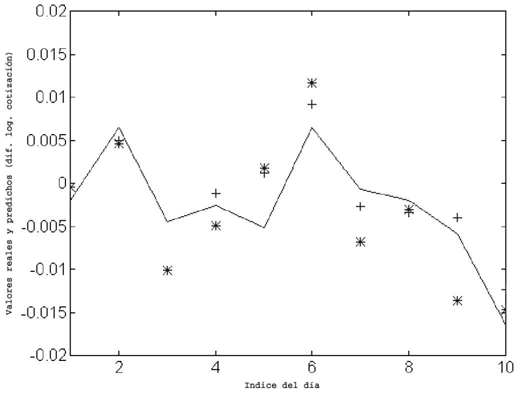

Figure 4: Prediction results with and without exogenous information using NAPA-PRED. The solid line is the real time series, asterisks (*) are results using NAPA-PRED, and crosses (+) are results using NAPA-PRED+PCA

Table 2: NRMSE for last 10 points (normalized round mean square error).

Mode	*NRMSE Error*
With PCA	0.7192
Without PCA	0.5189

mation extracted using this technique forces redefining the preprocessing step in a few iterations. This is related to the fact that we choose the principal components instead of the original series (5 variables); this choice would increase even more the input space dimension reducing the efficiency of the model.

3.4.2 Prediction Results

We compare prediction results obtained using the algorithm in section 2, rejecting the regularization term with and without the method proposed in this Section as is shown in figure 4. Prediction results improve and it is due to fraction of exogenous information included. In table 2 we show that (after transformations are inverted) the results for these 10 point are improved in a percentage around 7%.

3.5 Conclusions

From the results in the previous Section, itt's clear that the originally proposed method in [3], improves algorithms efficiency. This increase is based on selecting a suitable input space using a statistic method (PCA) and to date, it's the only way to develop it.

However the reader can notice the problems of this method. These problems are mentioned in the introduction of the chapter and are about the order of statistics used. In addition the increasing dimensionality ("curse of dimensionality") can cause serious problems (we were using a 5 dimensional input space) damaging the quality of the result. Neural networks are very sensitive to this problem because of the number of neurons to ensure universal approximation conditions [39] grows exponentially with the input space dimension unlike multilayer perceptrons, i.e. they are global approximations of nonlinear transformations, so they have a natural capacity of generalization [22, Sec. 7.9] with limited data set.

4 RNs and ICA

In this Section we describe a method for volatile time series forecasting using Independent Component Analysis (ICA) algorithms (see [40]) and Savitzky-Golay filtering as preprocessing tools. The preprocessed data will be introduce in a based radial basis functions (RBF) Artificial Neural Network (ANN) and the prediction result will be compared with the one we get without these preprocessing tools. This method is a generalization of the classical Principal Component Analysis (PCA) method for exogenous information inclusion (see Section 3)

4.1 Basic ICA

ICA has been used as a solution of the blind source separation problem [10] denoting the process of taking a set of measured signal in a vector, $\mathbf{x}$, and extracting from them a new set of statistically independent components (ICs) in a vector $\mathbf{y}$. In the basic ICA each component of the vector $\mathbf{x}$ is a linear instantaneous mixture of independent source signals in a vector $\mathbf{s}$ with some unknown deterministic mixing coefficients:

$$x_i = \sum_{i=1}^{N} a_{ij} s_j \tag{28}$$

Due to the nature of the mixing model we are able to estimate the original sources $\tilde{s}_i$ and the unmixing weights b_{ij} applying i.e. ICA algorithms based on higher order statistics such as cumulants.

$$\tilde{s}_i = \sum_{i=1}^{N} b_{ij} x_j \tag{29}$$

Using vector-matrix notation and defining a time series vector $\mathbf{x} = (x_1, \ldots, x_n)^T$, $\mathbf{s}$, $\tilde{\mathbf{s}}$ and the matrix $\mathbf{A} = \{a_{ij}\}$ and $\mathbf{B} = \{b_{ij}\}$ we can write the overall process as:

$$\tilde{\mathbf{s}} = \mathbf{Bx} = \mathbf{BAs} = \mathbf{Gs} \tag{30}$$

where we define $\mathbf{G}$ as the overall transfer matrix. The estimated original sources will be, under some conditions included in Darmois-Skitovich theorem (chapter 1 in [41]), a permuted and scaled version of the original ones. Thus, in general, it is only possible to find $\mathbf{G}$ such that $\mathbf{G} = \mathbf{PD}$ where $\mathbf{P}$ is a permutation matrix and $\mathbf{D}$ is a diagonal scaling matrix.

This model (equation (28)) can be applied to the stock series where there are some underlying factors like seasonal variations or economic events that affect the stock time series simultaneously and can be assumed to be quite independent [42].

4.2 Preprocessing Time Series with ICA+Filtering

The main goal, in the preprocessing step, is to find non-volatile time series including exogenous information i.e. financial time series, easier to predict using ANNs based on RBFs. This is due to smoothed nature of the *kernel* functions used in regression over multidimensional domains [43]. We propose the following Preprocessing Steps

- After whitening the set of time series $\{x_i\}_{i=1}^n$(subtracting the mean of each time series and removing the second order statistic effect or covariance matrix diagonalization process—see Section 3 for further details)

- We apply an ICA algorithm to estimate the original sources s_i and the mixing matrix $\mathbf{A}$ in equation (28). Each IC has information of the stock set weighted by the components of the mixing matrix. In particular, we use an equivariant robust ICA algorithm based in cumulants (see [41] and [6]) however another choices can be taken instead, i.e. in [40]. The unmixing matrix is calculated according the following iteration:

$$\mathbf{B}^{(n+1)} = \mathbf{B}^{(n)} + \mu^{(n)}(\mathbf{C}^{1,\beta}_{\mathbf{s},\mathbf{s}}\mathbf{S}^{\beta}_{\mathbf{s}} - \mathbf{I})\mathbf{B}^{(n)} \quad (31)$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{C}^{1,\beta}_{\mathbf{s},\mathbf{s}}$ is the $\beta+1$ order cumulant of the sources (we chose $\beta = 3$ in simulations) , $\mathbf{S}^{\beta}_{\mathbf{s}} = diag(sign(diag(\mathbf{C}^{1,\beta}_{\mathbf{s},\mathbf{s}})))$ and $\mu^{(n)}$ is the step size.

Once convergence, which is related to cross-cumulants [4] absolute value, is reached, we estimate the mixing matrix inverting $\mathbf{B}$.

Generally, the ICs obtained from the stock returns reveal the following aspects [8]:

1. Only a few ICs contribute to most of the movements in the stock return.
2. Large amplitude transients in th dominant ICs contribute to the major level changes. The non-dominant components do not contribute significantly to level changes.
3. Small amplitude ICs contribute to the change in levels over short time scales, but over the whole period, there is little change in levels.

- Filtering.

1. We neglect non-relevant components in the mixing matrix $\mathbf{A}$ according to their absolute value. We consider the rows $\mathbf{A}_i$ in matrix $\mathbf{A}$ as vectors and calculate the mean Frobenius norm [5] of each one. Only the components bigger than mean Frobenius norm will be considered. This is the principal preprocessing step using PCA tool but in this case this is not enough.

$$\tilde{\mathbf{A}} = \mathbf{Z} \cdot \mathbf{A} \quad (32)$$

where $\{\mathbf{Z}\}_{ij} = [\{\mathbf{A}\}_{ij} > \frac{||\mathbf{A}_i||_{Fr}}{n}]$

2. We apply a low band pass filter to the ICs. We choose the well-adapted for data smoothing Savitsky-Golay smoothing filter [44] for two reasons: a)ours is a real-time application for which we must process a continuous data stream and wish to output filtered values at the same rate we receive raw data and b) the quantity of data to be processed is so large that we just can afford only a very small number of floating operations on each data point thus computational cost in frequency domain for high dimensional data is avoided even the modest-sized FFT (see in [40]). This filter is also called Least-Squares [45] or DISPO [46]. These filters derive from a particular formulation of the data smoothing problem in the time domain and their goal is to find filter coefficients c_n in the expression:

$$\bar{s}_i = \sum_{n=-n_L}^{n_R} c_n s_{i+n} \quad (33)$$

where $\{s_{i+n}\}$ represent the values for the ICs in a window of length $n_L + n_R + 1$ centered on i and $\tilde{s}_i$ is the filter output (the smoothed ICs), preserving higher moments [47].

For each point s_i we least-squares fit a m order polynomial for all $n_L + n_R + 1$ points in the moving window and then set $\tilde{s}_i$ to the value of that polynomial at position i. As shown in [47] there are a set of coefficients for which equation (33) accomplishes the process of polynomial least-squares fitting inside a moving window:

$$c_n = \{(\mathbf{M}^T \cdot \mathbf{M})^{-1}(\mathbf{M}^T \cdot \mathbf{e}_n)\}_0 = \\ = \sum_{j=0}^{m}\{(\mathbf{M}^T \cdot \mathbf{M})^{-1}\}_{0j} \cdot n^j$$

where $\{\mathbf{M}\}_{ij} = i^j$, $i = -n_L, \ldots, n_R$, $j = 0, \ldots, m$, and $\mathbf{e}_n$ is the unit vector with $-n_L < n < n_R$. Note that equation (34) implies that we need only one row of the inverse matrix (numerically we can get this by LU decomposition [47], with only a single backsubstitution).

[4] Fourth order cumulant between each pair of sources must equals zero. This is the essential condition of statistical independence as is shown in chapter 3 in [6].

[5] Given $x \in \mathcal{R}^n$, its Frobenius norm is $||\mathbf{x}||_{Fr} \equiv \sqrt{\sum_{i=1}^n x_i^2}$

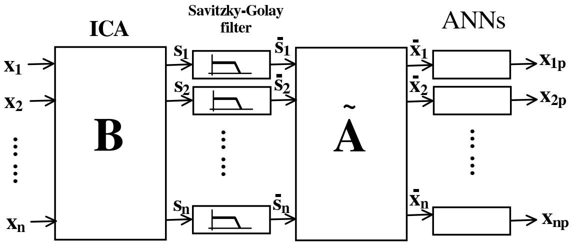

Figure 5: Schematic representation of prediction and filtering process.

- Reconstructing the original series using the smoothed ICs and filtered $\tilde{\mathbf{A}}$ matrix we obtain a less high frequency variance version of the series including exogenous influence of the exogenous ones. We can write using equations 42 and 41.

$$\mathbf{x} = \tilde{\mathbf{A}} \cdot \bar{\mathbf{s}} \tag{34}$$

4.3 Time Series Forecasting Model

We use an ANN based on RBFs to forecast a series x_i from the Stock Exchange building a forecasting function $\mathbf{P}$ with the help of the algorithm presented in Section 2, for one of the set of signals $\{x_1, \ldots, x_n\}$. As shown in Section 2 the individual forecasting function can be expressed in terms of RBFs as [48]:

$$\mathbf{f}(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x}) = \sum_{i=1}^{N} h_i \exp\{\frac{||\mathbf{x} - \mathbf{c}_i||^2}{\mathbf{r}_i^2}\} \tag{35}$$

where $\mathbf{x}$ is a p-dimensional vector input at time $\mathbf{t}$, $\mathbf{N}$ is the number of neurons (RBFs) , $\mathbf{f}_i$ is the output for each neuron i-th , $\mathbf{c}_i$ is the centers of i-th neuron which controls the situation of local space of this cell and $\mathbf{r}_i$ is the radius of the i-th neuron. The overall output is a linear combination of the individual output for each neuron with the weight of $\mathbf{h}_i$. Thus we are using a method for moving beyond the linearity where the core idea is to augment/replace the vector input $\mathbf{x}$ with additional variables, which are transformations of $\mathbf{x}$, and then use linear models in this new space of derived input features. RBFs are one of the most popular *kernel* methods for regression over the domain $\mathcal{R}^n$ and consist on fitting a different but simple model at each query point $\mathbf{c}_i$ using those observations close to this target point in order to get a smoothed function. This localization is achieved via a weighting function or *kernel* $\mathbf{f}_i$.

The preprocessing step suggested in Section 4.2 is necessary due to the dynamics of the series (the algorithm presented Section 2 is sensitive to this preprocessed series) and it will be shown that results improve noticeably [40]. Thus we use as input series the smoothed ones obtained from equation (45).

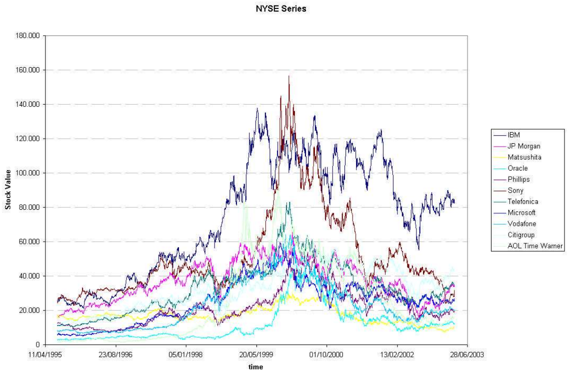

Figure 6: Set of stock series.

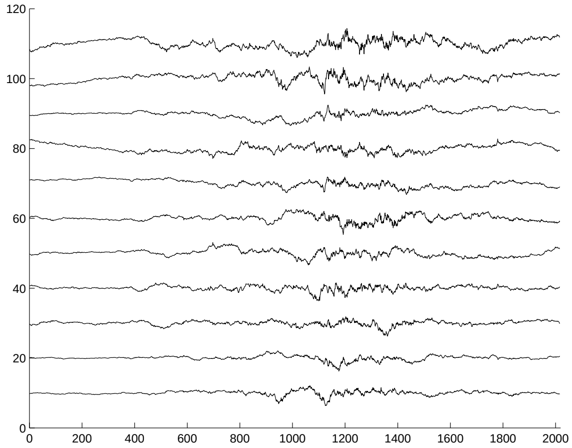

Figure 7: Set of ICs of the stock series.

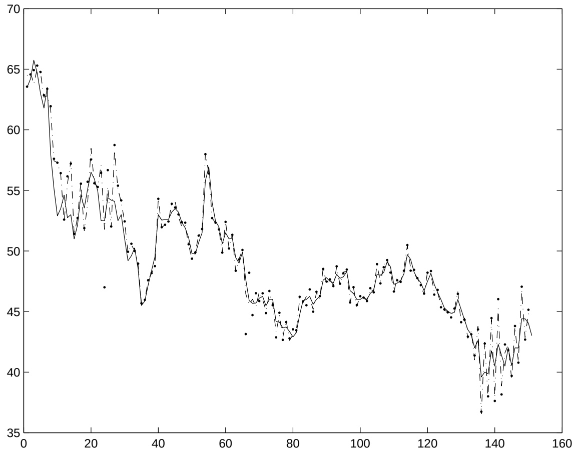

Figure 8: Real Series(line),predicted Series with ICA+SG(dash-dotted),predicted Series without preprocessing (dotted). The stock selected was Bakinter from IBEX35

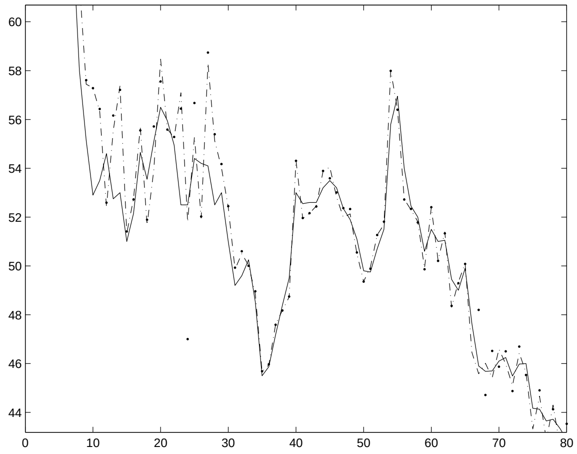

Figure 9: Zoom on figure 8.

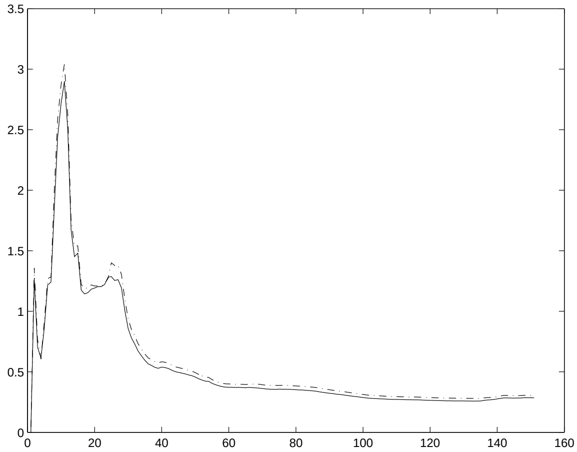

Figure 10: NRMSE evolution for ANN method and ANN+ICA method for Bankinter Series; theret's a noticeable improvement even under volatile conditions.

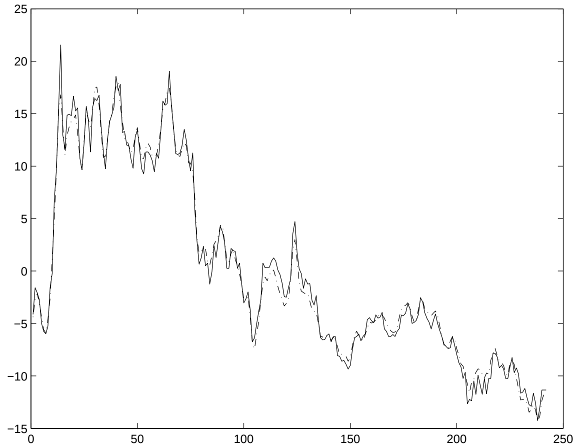

Figure 11: Real series from ICA reconstruction (scaled old version)(line) and preprocessed real series (dotted line). Selected Stock: Bankinter

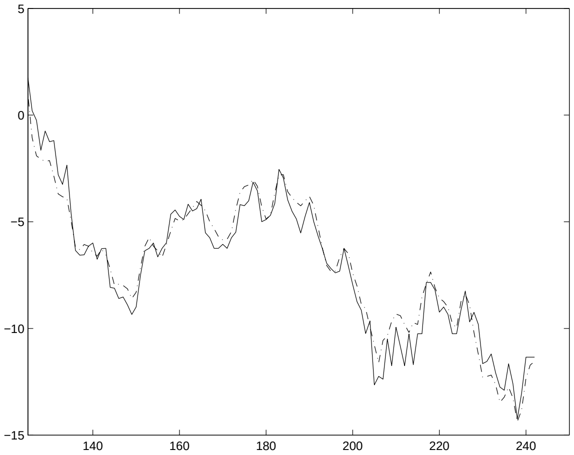

Figure 12: Zoom on figure 11.

4.4 Simulations

In the current simulation we have worked with an index of a Spanish bank (Bankinter) and other companies (such as exogenous variables) during the same period to investigate the effectiveness of ICA techniques for financial time series (figure 6). We have specifically focussed on the dowjones from american stock, which we consider the most representative sample of the american stock movements, using closing prices series.

We considered the closing prices of Bankinter for prediction and 10 indexes of intenational companies (IBM, JP Morgan, Matsushita, Oracle, Phillips, Sony, Microsoft, Vodaphone, Citigroup and Warner). Each time series includes 2000 points corresponding to selling days (quoting days).

We performed ICA on the Stock returns using the ICA algorithm presented in Section 4.2 assuming that the number of stocks equals the number of sources supplied to the mixing model. This algorithm whiten the raw data as the first step. The ICs are shown in the figure 7. These ICs represents independent and different underlying factors like seasonal variations or economic events that affect the stock time series simultaneously. Via the rows of $\mathbf{A}$ we can reconstruct the original signals with the help of these ICs i.e. Bankinter stock after we preprocess the raw data:

– Frobenius Filtering: the original mixing matrix[6]:

$$\mathbf{A} = \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots \\ \dots & 0.33 & -0.23 & 0.17 & 0.04 \\ \dots & \mathbf{-0.28} & \mathbf{1.95} & \mathbf{-0.33} & \mathbf{4.70} \\ \dots & 0.33 & -0.19 & 0.05 & -0.23 \\ \dots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \quad (36)$$

[6] We show and select the relevant part of the row corresponding to the Bankinter Stock.

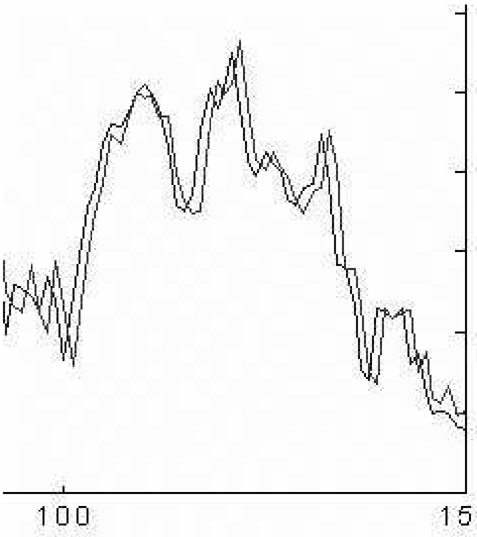

Figure 13: Delay problem in ANNs

is transformed to:

$$\tilde{\mathbf{A}} = \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots \\ \ldots & 0.33 & -0.23 & 0.17 & 0.04 \\ \ldots & \mathbf{0} & \mathbf{1.95} & \mathbf{0} & \mathbf{4.70} \\ \ldots & 0.33 & -0.19 & 0.05 & -0.23 \\ \ldots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \quad (37)$$

thus we neglect the influence of two ICs on the original 5^{th} stock. Thus only a few ICs contribute to most of the movements in the stock returns and each IC contributes to a level change depending its amplitude transient [8].

- We compute a polynomial fit in the ICs using the library supported by *MatLab* and the reconstruction of the selected stock (see figure 11) to supply the ANN.

In figures 8 and 9 we show the results (prediction for 150 samples) we obtained using our ANN with the latter ICA method. We can say that prediction is better with the preprocessing step avoiding the disturbing peaks or convergence problems in prediction. As is shown in figure 10, the NRMSE is always lower using the techniques we discussed in Section 4.3.

Finally, with these models we avoid the "curse of dimensionality" or difficulties associated with the feasibility of density estimation in many dimensions presented in AR or ANNs models (i.e RAN networks without input space control, see Section 2) with high number of inputs as shown in figure 14 and the delay problem presented in non relevant time periods of prediction (bad capacity of generalization in figure 13). In addition, we improve the model presented in Section 3 in two ways:

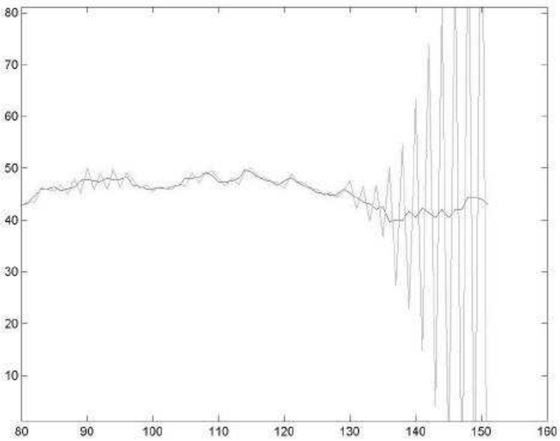

Figure 14: curse of dimensionality

- This method includes more relevant information (higher order statistics) in which PCA is the first step in the process.
- The way of including extra information avoids "curse of dimensionality" in any case.

4.5 Conclusions

In this Section we showed that prediction results can be improved with the help of techniques like ICA. ICA decompose a set of 11 returns from the stock into independent components which fall in two categories[40]:

1. Large components responsible of the major changes in level prices and
2. Small fluctuations responsible of undesirable fluctuations in time.

Smoothing this components and neglecting the non-relevant ones we can reconstruct a new version of the Stock easier to predict. Moreover we describe a new filtering method to volatile time series that are supplied to ANNs in real-time applications.

5 Multidimensional Regularization Networks

In this Section we propose the simplest way of including extra information(MRNs). We assume a linear model in some feature space on which the set of transformed input series will be fitted minimizing the empirical risk ("Feature Space Learning"). Various techniques can be applied to minimize this functional, i.e. we propose a genetic algorithm (GA) based on *neighbor philosophy* to speed up the convergence. This model, using the capacity of generalization of INAPA-PRED algorithm, is suitable for small sample size improving forecasting results under this condition.

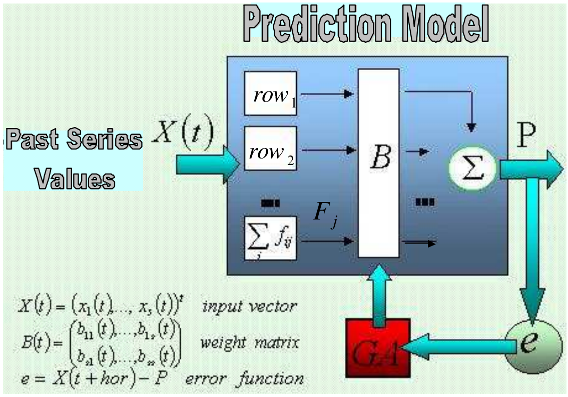
Prediction Model
Past Series Values
$X(t)$
row_1
row_2
$\sum_j f_{ij}$
F_j
B
Σ
P
GA
e
$X(t) = (x_1(t),\ldots,x_s(t))^t$ input vector
$B(t) = \begin{pmatrix} b_{11}(t),\ldots,b_{1s}(t) \\ b_{s1}(t),\ldots,b_{ss}(t) \end{pmatrix}$ weight matrix
$e = X(t+hor) - P$ error function

Figure 15: Schematic representation of MRN with adaptive radius, centers and input space ANNs (CPM CrossOver Prediction Model). [5]

5.1 Forecasting Model

The new prediction model is shown in figure 15. We consider a data set consisting of some correlated signals from the Stock Exchange and seek to build a forecasting function $\mathbf{P}$, for one of the sets of signals $\{series_1, \ldots, series_S\}$, which allows exogenous data from the other series to be included. If we consider just one series (see Section 2) the individual forecasting function can be expressed in terms of RBFs as in [48]:

$$\mathbf{F}(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x}) = \sum_{i=1}^{N} h_i \cdot \exp\left\{\frac{||\mathbf{x} - \mathbf{c}_i||^2}{\mathbf{r}_i^2}\right\} \quad (38)$$

where $\mathbf{x}$ is a p-dimensional vector input at time $\mathbf{t}$, $\mathbf{N}$ is the number of neurons (RBFs) , $\mathbf{f}_i$ is the output for each $i-th$ neuron, $\mathbf{c}_i$ is the centers of the $i-th$ neuron which controls the situation of local space of this cell and $\mathbf{r}_i$ is the radius of the i-th neuron. The overall output is a linear combination of the individual outputs for each neuron with a weight of $\mathbf{h}_i$. Thus we are using a method for moving beyond linearity in which the key idea is to augment/replace the vector input $\mathbf{x}$ with additional variables, which are transformations of $\mathbf{x}$, and then use linear models in this new space of derived input features. RBFs are one of the most popular *kernel* methods for regression over the domain $\mathcal{R}^n$ and consist of fitting a different but simple model at each query point $\mathbf{c}_i$ using the observations close to this target point in order to get a smoothed function (see previous Sections). This localization is achieved via a weighting function or *kernel* $\mathbf{f}_i$.

We apply/extend this regularization concept (see Section 2, to extra series, see figure15, including a row of neurons (equation (38)) for each series, and weight these values by a factor $\mathbf{b}_{ij}$. Finally, the overall smoothed function for the stock $\mathbf{j}$ is defined as:

$$\mathbf{P}_j(\mathbf{x}) = \sum_{i=1}^{S} \mathbf{b}_{ij} F_i(x_i, j) \quad (39)$$

where $\mathbf{F}_i$ is the smoothed function of each series, $\mathbf{S}$ is the number of input series and $\mathbf{b}_{ij}$ are the weights for j-stock forecasting. Obviously one of these weight factors must be relevant in this linear fit ($\mathbf{b}_{jj} \sim 1$, or auto weight factor).

Matrix notation can be used to include the set of forecasts in an S-dimensional vector $\mathbf{P}$ ($\mathbf{B}$ in figure15):

$$\mathbf{P}(\mathbf{x}) = diag(\mathbf{B} \cdot \mathbf{F}(\mathbf{x})) \quad (40)$$

where $\mathbf{F} = (\mathbf{F}_1, \ldots, \mathbf{F}_S)$ is an $S \times S$ matrix with $F_i \in \mathcal{R}^S$ and B is an $S \times S$ weight matrix. The operator $diag$ extracts the main diagonal. Because the number of neurons and the input space dimension increases in prediction function (equation (40)), we must control them (*parsimony*) to reduce curse of dimensionality effect and overfitting.

To check this model, we choose a set of values for the weight factors as functions of correlation factors between the series, and thus equation (39) can be expressed (replacing $\mathbf{P_j}$ with $\mathbf{P}$) as:

$$\mathbf{P}(\mathbf{x}) = (1 - \sum_{i \neq j}^{S} \rho_i)\mathbf{F}_j + \sum_{i \neq j}^{S} \rho_i \mathbf{F}_i \quad (41)$$

where $\mathbf{P}$ is the forecasting function for the desired stock j and $\boldsymbol{\rho}_i$ is the correlation factor with the exogenous series $\mathbf{i}$.

We can include equation (41) in the Generalized Additive models for regression proposed in supervised learning [43]:

$$\mathbf{E}\{Y|\mathbf{X}_1, \ldots, \mathbf{X}_n\} = \alpha + \mathbf{f}_1(\mathbf{X}_1) + \ldots + \mathbf{f}_n(\mathbf{X}_n) \quad (42)$$

where $\mathbf{X}_i s$ usually represent predictors and $\mathbf{Y}$ represents the system output; $\mathbf{f}_j s$ are unspecific smooth ("nonparametric") functions. Thus we can fit this model by minimizing the mean square error function or by other methods presented in [43] (in the next Section we use a GA, a well known optimization tool, to minimize the mean square error).

5.2 Forecasting Model and Genetic Algorithms

MRN uses a GA for $\mathbf{b}_i$ parameter fitting. A GA can be modelled by means of a *time inhomogeneous Markov* chain [49] obtaining interesting properties related to weak and strong ergodicity, convergence and the distribution probability of the process (see [50]). In the latter reference, a canonical GA is constituted by operations of parameter encoding, population initialization, crossover , mutation, mate selection, population replacement, fitness scaling, etc. proving that with these simple operators a GA does not converge to a population containing only optimal members. However, there are GAs that converge to the optimum, *The Elitist GA* [51] and those which introduce *Reduction Operators* [52].

We have borrowed the notation mainly from [53] where the model for GAs is a inhomogeneous Markov chain

Table 3: Pseudo-code of GA.

```
Initialize Population
i=0
while not stop do
  do N/2 times
    Select two mates from $\mathbf{p_i}$
    Generate two offspring using crossover operator
    Mutate the two children
    Include children in new generation $\mathbf{p_{new}}$
  end do
  Build population $\hat{\mathbf{p}}_\mathbf{i} = \mathbf{p_i} \cup \mathbf{p_{new}}$
  Apply Reduction Operators (Elitist Strategies) to get $\mathbf{p_{i+1}}$
  i=i+1
end
```

model on probability distributions ($\mathbf{S}$) over the set of all possible populations of a fixed finite size. Let $\mathbf{C}$ the set of all possible creatures in a given world (vectors of dimension equal to the number of extra series) and a function $f : \mathbf{C} \rightarrow R^+$. The task of GAs is to find an element $c \in \mathbf{C}$ for which $f(c)$ is maximal. We encode creatures into genes and chromosomes or individuals as strings of length ℓ of binary digits (size of Alphabet A is $a = 2$) using one-complement representation; other encoding methods, also possible i.e [54], [55],[56] or [57], where the value of each parameter is a gene and an individual is encoded by a string of real numbers instead of binary ones.

In the Initial Population Generation step (choosing randomly $p \in \wp_N$, where $\wp_N$ is the set of populations, i.e the set of N-tuples of creatures containing $a^{L \equiv N \cdot \ell}$ elements) we assume that creatures lie in a bounded region $[0, 1]$ (at the edge of this region we can reconstruct the model without exogenous data). After the initial population p has been generated, the fitness of each chromosome $\mathbf{c}_i$ is determined via the function:

$$f(c_i) = \frac{1}{\mathbf{e}(\mathbf{c}_i)} \tag{43}$$

where $\mathbf{e}$ is an error function (i.e square error sum in a set of neural outputs, adjusting the convergence problem in the optimal solution by adding a positive constant to the denominator)

The next step in canonical GA is to define the Selection Operator. New generations for mating are selected depending on their fitness function values using *roulette wheel selection*. Let $p = (c_1, \ldots, c_N) \in \wp_N$, $n \in \mathcal{N}$ and f the fitness function acting in each component of p. Scaled fitness selection of p is a lottery for every position $1 \leq i \leq N$ in population p such that creature c_j is selected with probability:

$$\frac{f_n(p, j)}{\sum_{i=1}^{N} f_n(p, i)} \tag{44}$$

thus proportional fitness selection can be described by column stochastic matrices $\mathbf{F_n}$, $n \in \mathcal{N}$, with components:

$$\langle q, \mathbf{F_n} p \rangle = \prod_{i=1}^{N} \frac{n(q_i) f_n(p, q_i)}{\sum_{j=1}^{N} f_n(p, j)} \tag{45}$$

where $p, q \in \wp_N$ so $p_i, q_i \in \mathbf{C}$, $\langle \ldots \rangle$ denotes the standard inner product, and $n(d_i)$ the number of occurrences of q_i in p.

Once the two individuals have been selected, an elementary crossover operator $\mathbf{C}(K, P_c)$ is applied (setting the crossover rate at a value, i.e. $P_c \rightarrow 0$, which implies children similar to parent individuals) that is given (assuming N even) by:

$$\mathbf{C}(K, P_c) = \prod_{i=1}^{N/2} ((1 - P_c)\mathcal{I} + P_c \mathbf{C}(2i - 1, 2i, k_i)) \tag{46}$$

where $\mathbf{C}(2i - 1, 2i, k_i)$ denotes elementary crossover operation of c_i, c_j creatures at position $1 \leq k \leq \ell$ and $\mathcal{I}$ the identity matrix, to generate two offspring (see [50] for details of the crossover operator).

The Mutation Operator $\mathbf{M_{P_m}}$ is applied (with probability $\mathbf{P}_m$) independently at each bit in a population $p \in \wp_N$, to avoid premature convergence (see [54] for further discussion). The multi-bit mutation operator with change probability following a *simulated annealing* law with respect to the position $1 \leq i \leq L$ in $p \in \wp_N$:

$$P_m(i) = \mu \cdot \exp\left(\frac{-mod\{\frac{i-1}{N}\}}{\emptyset}\right) \tag{47}$$

where $\emptyset$ is a normalization constant and μ the change probability at the beginning of each creature p_i in population p; can be described as a positive stochastic matrix in the form:

$$\begin{gathered}\langle q, \mathbf{M_{P_m}} p \rangle = \mu^{\Delta(p,q)} \exp\left(-\textstyle\sum_{dif(i)}^{\Delta(p,q)} \frac{mod\{\frac{i-1}{N}\}}{\emptyset}\right) \\ \cdot \textstyle\prod_{equ(i)}^{L-\Delta(p,q)} \left[1 - \mu \cdot \exp\left(\frac{-mod\{\frac{i-1}{N}\}}{\emptyset}\right)\right]\end{gathered} \tag{48}$$

where $\Delta(p, q)$ is the Hamming distance between p and $q \in \wp_N$, $dif(i)$ resp. $equ(i)$ is the set of indexes where p and q are different resp. equal. Following from equation (48) and checking how the matrices act on populations we can write:

$$\mathbf{M_{P_m}} = \prod_{\lambda=1}^{N} \left([1 - P_m(\lambda)]\, \mathbf{1} + P_m(\lambda) \hat{\mathbf{m}}^1(\lambda)\right) \tag{49}$$

where $\hat{\mathbf{m}}^1(\lambda) = \mathbf{1} \otimes \mathbf{1} \ldots \otimes \overbrace{\hat{m}^1}^{\lambda} \otimes \ldots \otimes \mathbf{1}$ is a linear operator on $V_\wp$, the free vector space over A^L and $\hat{m}^1$ is the linear 1-bit mutation operator on V_1, the free vector space over A. The latter operator is defined acting on Alphabet as:

$$\langle \hat{a}(\tau'), \hat{m}^1 \hat{a}(\tau) \rangle = (a-1)^{-1}, \quad 0 \le \tau' \ne \tau \le a-1 \quad (50)$$

i.e. probability of change a letter in the Alphabet once mutation occurs with probability equal to $L\mu$.

The spectrum of $\mathbf{M_{P_m}}$ can be evaluated according to the following expression:

$$sp(\mathbf{M_{P_m}}) = \left\{ \left(1 - \frac{\mu(\lambda)}{a-1}\right)^{\lambda}; \quad \lambda \in [0, L] \right\} \quad (51)$$

where $\mu(\lambda) = \exp\left(\frac{-mod\{\frac{\lambda-1}{N}\}}{\emptyset}\right)$.

The operator presented in equation (49) has similar properties to the Constant Multiple-bit mutation operator $\mathbf{M}_{\boldsymbol{\mu}}$. $\mathbf{M}_{\boldsymbol{\mu}}$ is a contracting map in the sense presented in [53]. It is easy to prove that $\mathbf{M_{P_m}}$ is a another contracting map, using the Corollary B.1 in [6] and the eigenvalues of this operator(equation (51)).

We can also compare the coefficients of ergodicity:

$$\tau_r(\mathbf{M_{P_m}}) < \tau_r(\mathbf{M}_{\boldsymbol{\mu}}) \quad (52)$$

where $\tau_r(\mathbf{X}) = max\{\|\mathbf{X}v\|_r : v \in \mathcal{R}^n, \quad v \perp e \quad and \quad \|v\|_r = 1\}$.

Mutation is more likely at the beginning of the string of binary digits ("small neighborhood philosophy"). In order to improve the speed convergence of the algorithm we have included mechanisms such as elitist strategy (reduction operator [58]) in which the best individual in the current generation always survives into the next (a further discussion about reduction operator, $\mathbf{P}_R$, can be found in [59]).

Finally the GA is modelled, at each step, as the stochastic matrix product acting on probability distributions over populations:

$$\mathbf{S_{P^n_m, P^n_c}} = \mathbf{P}^n_R \cdot \mathbf{F_n} \cdot \mathbf{C^k_{P^n_c}} \cdot \mathbf{M_{P^n_m}} \quad (53)$$

The GA used in forecasting function (equation (39)) has absolute error value start criterion (i.e $error > uga = 1.5$). Once it starts, it uses the values (or individual) found to be optimal (elite) the last time, and applies local search (using the selected mutation and crossover operators) around this elite individual. Thus we perform an efficient search around an individual (set of $\mathbf{b}_i s$) in which one parameter is more relevant than the others.

The computational time depends on the encoding length, number of individuals and genes. Because of the probabilistic nature of the GA- based method, the proposed method almost converges to a global optimal solution on average. In our simulation nonconvergent case was found. Table 3 shows the GA-pseudocode and in [5] the iterative procedure implemented for the overall prediction system including GA is shown.

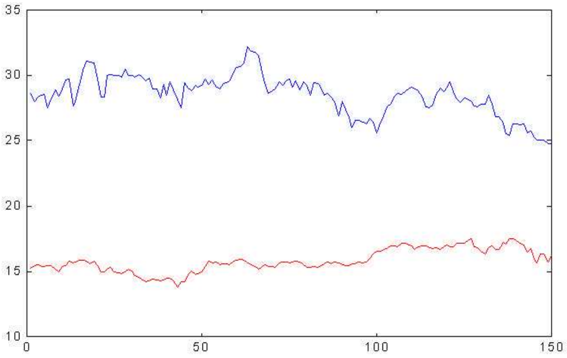

Figure 16: Set of data series. Top: Real Series ACS;Bottom: Real Series BBVA.

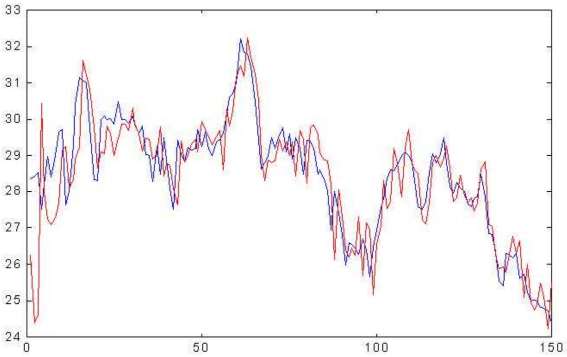

Figure 17: Real Series and Predicted ACS Series with MRN.

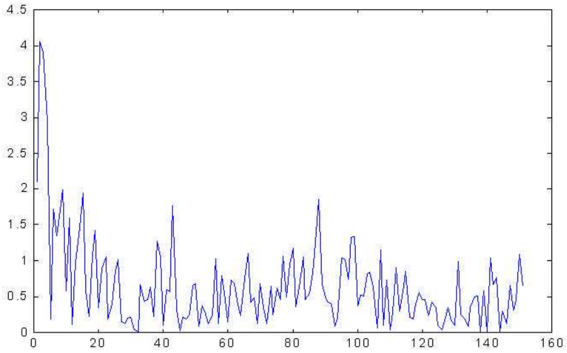

Figure 18: Absolute Error Value with MRN.

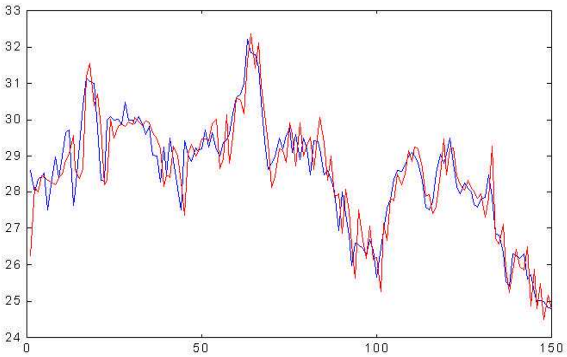

Figure 19: Real Series and Predicted ACS Series with MRN+GA.

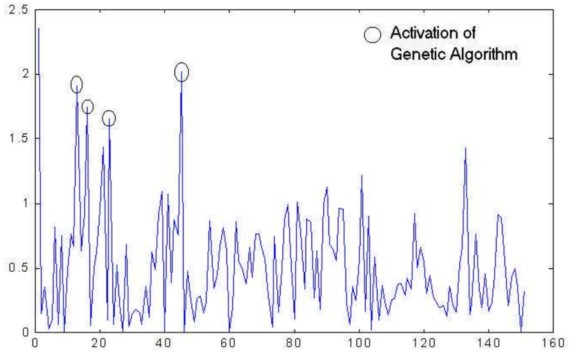

Figure 20: Absolute Error Value with MRN + GA.

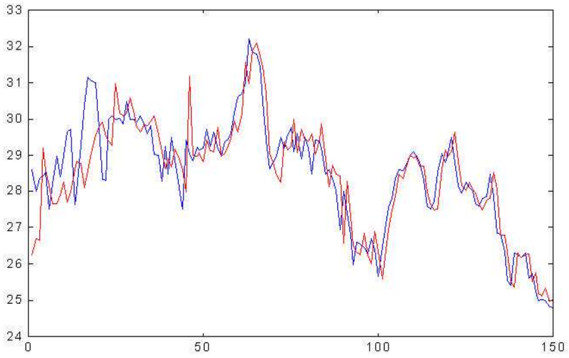

Figure 21: Real Series and Predicted ACS Series without exogenous data.

5.3 Simulations and Conclusions.

With the aim of assessing the performance of the MRN we have worked with indexes of different Spanish banks and other companies during the same period. We have specifically focussed on the IBEX35 index of Spanish stock, which we consider the most representative sample of Spanish stock movements. We used *MatLab* to implement MRN on a Pentium III at 850MHz.

We started by considering the most simplest case, which consists of two time series corresponding to the companies ACS ($series_1$) and BBVA ($series_2$). The first one is the target of the forecasting process; the second one is introduced as external information. The period under study covers the year 2000. Each time series includes 200 points corresponding to selling days (quoting days).

We highlight two parameters in the simulation process. The horizon of the forecasting process (*hor*) was set at 1; the weight function of the forecasting function was a correlation function between the two time series for the $series_2$ (in particular we chose its square) and the difference to one for the series 1. We took a forecasting window (**W**) of 10 lags, and the maximum lag number was set at double the value of W, and thus we built a 10×20 Toeplitz matrix. We started at time point $t_o = 50$. Figures 17,18 19 and 20 show the forecasting results from lag 50 to lag 200 corresponding to $series_1$.

Note the instability of the system in the very first itera-

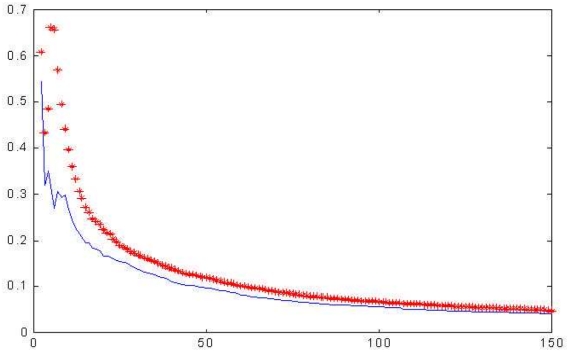

Figure 22: NRMSE evolution for MRN(dot) MRN + GA(line).

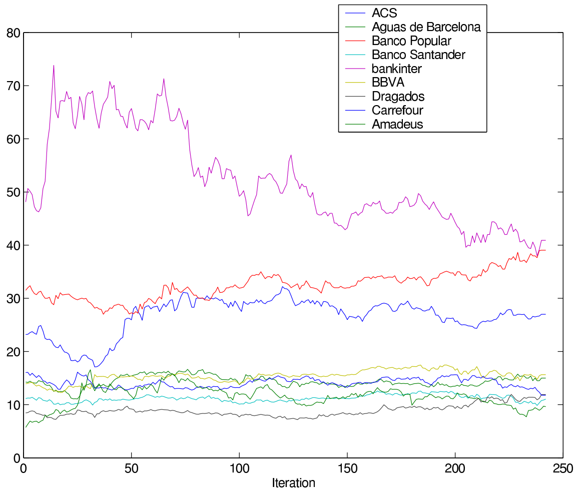

Figure 23: Set of series for complete simulation.

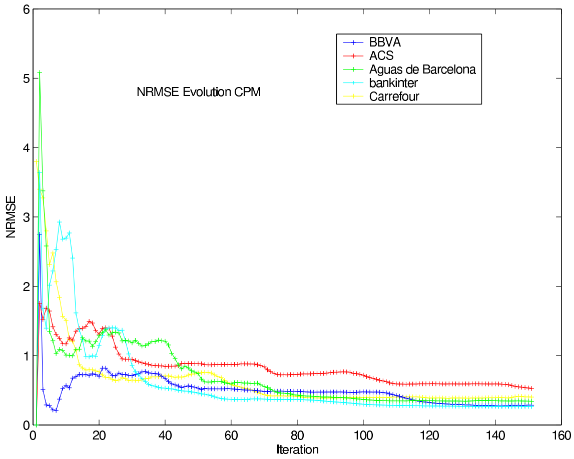

Figure 24: NRMSE evolution for selected stock indexes.

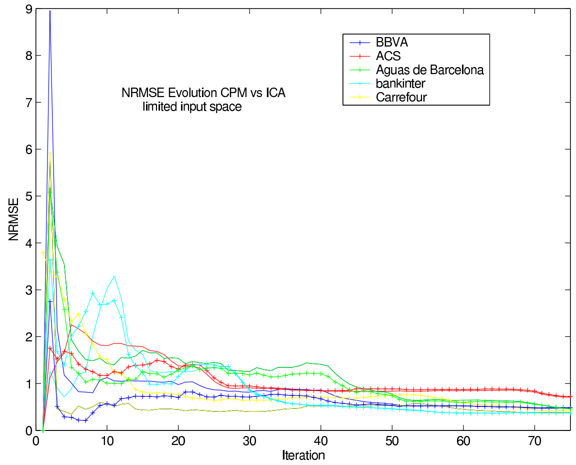

Figure 25: NRMSE evolution using ICA method and MRN.

Table 4: Correlation coefficients between real signal and the predicted signal for different lags.

delay	ρ	delay	ρ
0	0.89	0	0.89
+1	0.79	-1	0.88
+2	0.73	-2	0.88
+3	0.68	-3	0.82
+4	0.63	-4	0.76
+5	0.59	-5	0.71
+6	0.55	-6	0.66
+7	0.49	-7	0.63
+8	0.45	-8	0.61
+9	0.45	-9	0.58
+10	0.44	-10	0.51

Table 5: Dynamics and values of the weights for the GA.

b_{series}	T_1	T_2	T_3	T_4
b_1	0.8924	0.8846	0.8723	0.8760
b_2	0.2770	0.2359	0.2860	0.2634

tions until it reaches an acceptable convergence. The most interesting feature of the result is shown in table 4; from this table it is easy to deduce that if we move one of the two series horizontally the correlation between them dramatically decreases. This proves that we avoid the delay problem (trivial prediction) shown by certain networks (see figure 21), in periods where the information introduced to the system is non-relevant. This is due to the increase of information ($series_2$) associated with an increase in neuron resources. At the end of the process we used 20 neurons for net 1 and 21 for net 2. Although forecasting function is acceptable we would expect a better performance with bigger data set.

The next step consists of using the complete algorithm including the GA. A population of 40 individuals (N_{ind}) was used, with a 2×1 dimension; we used this small number because we had a bounded searching space and we were using a single PC. The genetic algorithm was run four times before reaching the convergence (when the error increase by 1.5 points, see error plot in figure 20 to see the effect of GA) ; the individuals were codified with 34 bits (17 bits for each parameter). In this case convergence is defined in terms of the adjustment function; other authors use other parameters of the GA, like the absence of change in the individuals after a certain number of generations, etc. We observed a considerable improvement in the forecasting results and noted disappearance of the delay problem, as is shown in table 4. This table represents the correlation between the real function and the neural function for different lags. The correlation function presents a maximum at lag 0.

The number of neurons at the end of the process is the same as in the latter case, because we have only modified the weight of each series during the forecasting process. The dynamics and values of the weights are shown in table 5.

Error behaviour is shown in figures. Note:

– We can bound the error by means of a suitable selection of the parameters $\mathbf{b_i}$, when the dynamics of the series is coherent (avoiding large fluctuations in the stock).

– The algorithm converges faster, as is shown at the very beginning of the graph.

– The forecasting results are better using GA, as is shown in figure 20, where the evolution of the normalized round mean square error is plotted.

Finally we carried out a simulation with 9 indexes and computed the prediction function for 5 series, obtaining similar results, as presented in figure 24. In the complete model we limited the input space dimension to 3 for extra series and 5 for target series (figure 22). NRMSE depends on each series (data set) and target series (evolution). In figure 25 we also compare the ICA method versus MRN for limited data set (70 iterations). NRMSE of indexes increases at the beginning of the process using the ICA method and converges to CPM NRMSE values when the data set increases (estimators approach higher order statistics). This effect is also observed when the dynamics of the series change suddenly due to a new independent event.

Due to the symmetric character of our forecasting model, it is sufficient to implement it in parallel programming software (such as PVM —*Parallel Virtual Machine*—) or MPI — *Message-Passing Interface*— [60]) to build a more general forecasting model for the complete set of series. We would spawn the same number for offspring processes and banks; these process would run forecasting vectors, which would be weighted by a square matrix with dimension equal to the number of series $\mathbf{B}$. The “master” process would have the results of the forecasting process for the calculus of the error vector, in order to update the neuron resources. Thus we would take advantage of the computational cost of a forecasting function to calculate the rest of the series (see [60]).

5.3.1 Conclusions

This new forecasting model for time-series is characterized by:

- The enclosing of external information. We avoid preprocessing and data contamination applying by ICA and PCA for limited data sets or sudden new shocks. These techniques can be included in MRN under better conditions. Series are introduced into the net directly.
- The forecasting results are improved using hybrid techniques like GA.
- The possibility of implementing in parallel programming languages (i.e. PVM — see [60]); and the improved performance and lower computational time achieved using a parallel neural network.

6 Comparison among methods

Consider the set of series in Figure 6, using 1000 point as training samples. We apply the latter models to forecast Sony index in 70 future points using the other indexes as endogenous variables. Following the methodology described in the previous sections we get:

- PCA operation: in this case results using PCA are not so good as we expected. The volatile nature of the series and the lack of information from the 3 principal components used contributes to this failure. The 3 components only hold a fraction equal to 64% of variance as is shown in Table 6, however if we increase the number of components used in prediction we get worse results owing to “curse of dimensionality”.
- MRN operation: MRN get good prediction results comparing with PCA method, however in the last iterations PCA and MRN methods are of similar NRMSE. The main disadvantages of MRN are computational demand and the need for bounding input space dimension.
- ICA operation: ICA is the best method using large sample size as is shown in Figure 28. ICA reveals some underlying structure in the data since we used HOS to estimate ICs that usually fall into two categories as we mentioned in section 4(see Figure 27): infrequent but large shocks (responsible for the major changes in the stock prices) and frequent but rather small fluctuations (contributing only little to the overall level of the stocks).

Table 6: Eigenvalues λ_i and variance percentages of PCs.

Index	*Eigenvalue*	*Pct. variance*	*Pct. overall*
1	0.0031	35.1642	35.1642
2	0.0016	17.9489	53.1131
3	0.0010	11.7188	64.8319
4	0.0006	6.9857	71.8176
5	0.0005	5.7787	77.5963
6	0.0004	4.8978	82.4941
7	0.0004	4.4184	86.9125
8	0.0004	4.2492	91.1617
9	0.0003	3.7238	94.8855
10	0.0003	3.2031	98.0886
11	0.0002	1.9113	100.00

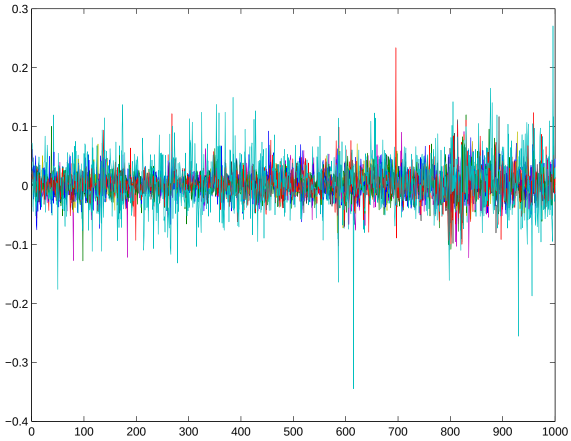

Figure 26: Closing prices evolution for selected indexes (Figure 6), after logarithmic transformation and differentiation.

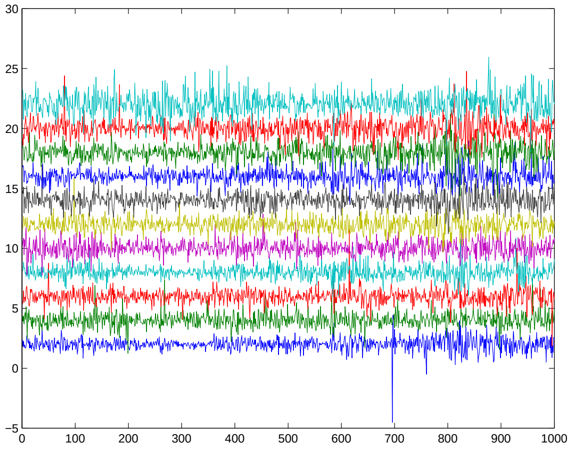

Figure 27: ICA ICs for the set of indexes.

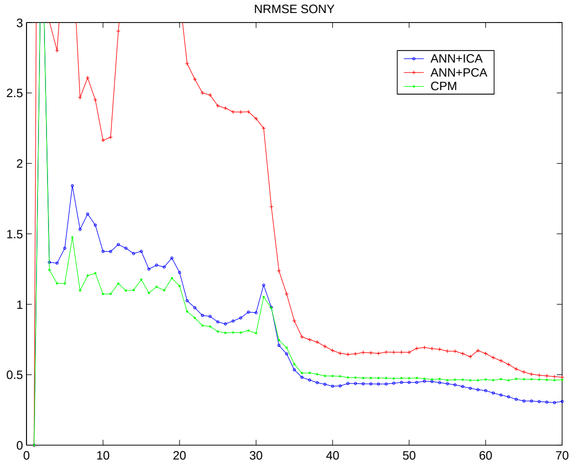

Figure 28: NRMSE evolution of SONY index for the 70 forecasted points using exogenous methods.

Acknowledgement

We want to thank SESIBON and HIWIRE grants from Spanish Government for funding.

References

[1] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting and Control*, 3rd ed. Prentice-Hall. Englewood Cliffs, New Jersey, U.S.A., 1994.

[2] J. Platt, "A resource-allocating network for function interpolation," *Neural Computation*, vol. 3, pp. 213–225, 1991.

[3] M. Salmerón, J. Ortega, C. Puntonet, and F. Pelayo, *Time Series Prediction with Hybrid Neuronal, Statistical and Matrix Methods (in Spanish)*. Department of Computer Architecture and Computer Technology. University of Granada, Spain, 2001.

[4] M. Salmerón, J. Ortega, C. G. Puntonet, and A. Prieto, "Improved ran sequential prediction using orthogonal techniques," *Neurocomputing*, vol. 41, pp. 153–172, 2001.

[5] J. M. Górriz, C. G. Puntonet, M. Salmerón, and J. González, "New model for time-series forecasting using rbfś and exogenous data," *Neural Computation and Applications, Vol 13 Issue 2. Jun. 2004.*, 2003.

[6] J. M. Górriz, "Algoritmos híbridos para la modelización de series temporales con técnicas ar-ica," Ph.D. dissertation, University of Cádiz , Departamento de Ing. de Sistemas y Aut. Tec. Electrónica y Electrónica. http://wwwlib.umi.com/cr/uca/main, 2003.

[7] T. Masters, *Neural, Novel and Hybrid Algorithms for Time Series Prediction*. John Wiley & Sons. New York, U.S.A., 1995.

[8] A. Back and A. Weigend, "Discovering structure in finance using independent component analysis," *Computational Finance*, 1997.

[9] A. Back and T. Trappenberg, "Selecting inputs for modelling using normalized higher order statistics and independent component analysis," *IEEE Transactions on Neural Networks*, vol. 12, 2001.

[10] A. Hyvarinen and E. Oja, "Independent component analysis: Algorithms and applications," *Neural Networks*, vol. 13, pp. 411–430, 2000.

[11] A. Bell and T. Sejnowski, "An information-maximization approach to blind separation and blind deconvolution," *Neural Computation*, vol. 7, pp. 1129–1159, 1995.

[12] P. Comon, “Independent component analysis, a new concept?” *Signal Processing*, vol. 3, pp. 287–314, 1994.

[13] S. Amari, A. Cichocki, and H. Yang, “A new learning algorithm for blind source separation,” *Advances in Neural Information Processing Systems. MIT Press*, vol. 8, pp. 757–763, 1996.

[14] C. G. Puntonet, *Nuevos algoritmos de Separación de fuentes en medios Lineales (in Spanish)*. Department of Computer Architecture and Computer Technology. University of Granada, Spain, 1994.

[15] A. Mansour, N. Ohnishi, and C. Puntonet, “Blind multiuser separation of instantaneous mixture algorithm based on geometrical concepts,” *Signal Processing*, vol. 82, pp. 1155–1175, 2002.

[16] D. Pollock, *A Handbook of Time Series Analysis, Signal Processing and Dynamics*. Academic Press. New York, U.S.A., 1999.

[17] G. Zhang, B. Patuwo, and M. Hu, “Forecasting with artificial neural networks: the state of the art,” *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[18] J. Moody and C. J. Darken, “Fast learning in networks of locally-tuned processing units,” *Neural Computation*, vol. 1, pp. 284–294, 1989.

[19] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, L. L. Cam and J. Neyman, Eds., vol. 1. Berkeley, California, U.S.A.: University of California Press, 1967, pp. 281–297.

[20] C. Lawson and R. Hanson, *Solving Least Squares Problems*. SIAM Publications. Philadelphia, U.S.A., 1995.

[21] G. Zelniker and F. Taylor, *Advanced Digital Signal Processing: Theory and Applications*. Marcel Dekker. New York, U.S.A., 1994.

[22] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing Company. New York, U.S.A., 1994.

[23] A. Tikhonov and V. Arsenin, *Solutions of Ill-Posed Problems*. Winston. Washington D.C., U.S.A., 1977.

[24] J. Mercer, “Functions of positive and negative type and their connection with the theory of integral equations,” *Philos. Trans. Roy. Soc. London*, vol. A, pp. 415–446, 1909.

[25] A. J. Smola, B. Schölkopf, and K.-R. Müller, “The connection between regularization operators and support vector kernels,” *Neural Networks*, vol. 11, pp. 637–649, 1998.

[26] J. M. Górriz, C. G., Puntonet, and M. Salmerón, “On line algorithm for time series prediction based on support vector machine philosophy,” *LNCS*, vol. 3037, pp. 50–57, 2004.

[27] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” *J. Chem. Phys.*, vol. 21, pp. 1087–1092, 1953.

[28] T. Kohonen, “The self-organizing map,” *Proceedings of the IEEE*, vol. 78, pp. 1464–1480, 1990.

[29] L. Cao, “Support vector machines experts for time series forecasting,” *Neurocomputing*, pp. 321–339, 2003.

[30] A. Comrey, *A First Course in Factor Analysis*. Academic Press. New York, U.S.A., 1973.

[31] J. Ramsay and B. Silverman, *Functional Data Analysis*. Springer-Verlag. New York, U.S.A., 1997.

[32] R. J. M. II, “Intelligence: Computational versus artificial,” *IEEE Transactions on Neural Networks*, vol. 4, pp. 737–739, 1993.

[33] J. Jackson, *A User’s Guide to Principal Components*. John Wiley & Sons. New York, U.S.A., 1991.

[34] G. Golub and C. V. Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press. Baltimore, Maryland, U.S.A., 1996.

[35] J. Hertz, A. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation*. Addison-Wesley. Redwood City, California, U.S.A., 1991.

[36] K. Diamantaras and S. Kung, *Principal Component Neural Networks: Theory and Applications*. John Wiley & Sons. New York, U.S.A., 1996.

[37] E. Oja, “Principal components, minor components, and linear neural networks,” *Neural Networks*, vol. 5, pp. 927–935, 1992.

[38] W. Sarle, “Neural networks and statistical models,” in *Proceedings of the 19th Anual SAS Users Group International Conference*, Cary, NC, U.S.A., 1994, pp. 1538–1550.

[39] J. Park and I. Sandberg, “Universal approximation using radial-basis-function networks,” *Neural Computation*, vol. 3, pp. 246–257, 1991.

[40] J. M. Górriz, C. G. Puntonet, M. Salmerón, and J. Ortega, “New method for filtered ica signals applied to volatile time series,” *7th International Work Conference on Artificial and Natural Neural Networks IWANN 2003. Lecture Notes in Computer Science Vol 2687 / 2003, Springer 433-440*, 2003.

[41] S. Cruces, *An unified view of BSS algorithms(in Spanish).* University of Vigo, Spain, 1999.

[42] K. Kiviluoto and E. Oja, "Independent component analysis for parallel financial time series," *Proc. in ICONIP98*, vol. 1, pp. 895–898, 1998.

[43] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of Statistical Learning*. Springer, 2000.

[44] A. Savitzky and M. Golay, *Analytical Chemestry*, vol. 36, pp. 1627–1639, 1964.

[45] R. Hamming, *Digital Filters*, 2^a ed. Prentice Hall, 1983.

[46] H. Ziegler, *Applied Spectroscopy*, vol. 35, pp. 88–92, 1981.

[47] W. Press, S. Teukolsky, W. Vertterling, and B. Flannery, *Numerical Recipes in C++*, 2^a ed. Cambridge University Press, 2002.

[48] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281–294, 1989.

[49] O. Haggstrom, *Finite Markov Chains and Algorithmic Applications.* Cambridge University, 1998.

[50] L. Schmitt, C. Nehaniv, and R. Fujii, "Linear analysis of genetic algorithms," *Theoretical Computer Science*, vol. 200, pp. 101–134, 1998.

[51] J. Suzuki, "A markov chain analysis on simple genetic algorithms," *IEEE Transaction on Systems, Man, and Cybernetics*, vol. 25, pp. 655–659, 1995.

[52] A. Eiben, E. Aarts, and K. V. Hee, "Global convergence of genetic algorithms: a markov chain analysis," *Parallel Problem Solving from Nature, Lecture Notes in Computer Science*, vol. 496, pp. 4–12, 1991.

[53] L. Schmitt, "Theory of genetic algorithms," *Theoretical Computer Science*, vol. 259, pp. 1–61, 2001.

[54] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1992.

[55] T. S. S. Matwin and K. Haigh, "Genetic algorithms approach to a negotiation support system," *IEEE Trans. Syst. , Man. Cybern*, vol. 21, pp. 102–114, 1991.

[56] S. Chen and Y. Wu, "Genetic algorithm optimization for blind channel identification with higher order cumulant fitting," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 259–264, 1997.

[57] L. Chao and W. Sethares, "Non linear parameter estimation via the genetic algorithm," *IEEE Transactions on Signal Processing*, vol. 42, pp. 927–935, 1994.

[58] J. Lozano, P. Larranaga, M. Grana, and F. Albizuri, "Genetic algorithms: Bridging the convergence gap," *Theoretical Computer Science*, vol. 229, pp. 11–22, 1999.

[59] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Transactions on Neural Networks*, vol. 5, pp. 96–101, 1994.

[60] J. M. Górriz, C. G. Puntonet, M. Salmerón, and R. Martin-Clemente, "Parallelization of time series forecasting model," *12 th IEEE Conference on Parallel, Distributed and Network based Processing (Euromicro) PDP 2004 pp 103-112. A Coruña, Spain.*, 2004.