Ph.D. in Electronic and Computer Engineering
Dept. of Electrical and Electronic Engineering
University of Cagliari

Inference Engines for Streaming Datasets

Marco STOCCHI

Advisor: Prof. Michele MARCHESI
Curriculum: ING-INF/05

XXIX Cycle
November 2016

Ph.D. in Electronic and Computer Engineering
Dept. of Electrical and Electronic Engineering
University of Cagliari

Inference Engines for Streaming Datasets

Marco STOCCHI

Advisor: Prof. Michele MARCHESI
Curriculum: ING-INF/05

XXIX Cycle
November 2016

Dedicated to a once beloved person

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The problem of forecasting streaming datasets, particularly the financial time series, has been largely explored in the past, but we believe the advancement of technologies such as the Internet of Things, which will connect an exponentially increasing number of sensors and devices, endowed with limited computational resources, yet capable of producing enormous amounts of sampled data, and the progressively higher social need to deploy intelligent systems, will make the prediction of time series a core industrial issue in the next future. Consequently, we also believe that investigating efficient models for accurate and reliable forecasting can be considered an urgent area of research.

Considerable efforts have been dedicated in the past in the attempt to forecast the foreign exchange rates, because of their typical non linearities and the associated well known difficulty of predicting such financial series. As such, they were historically considered one of the most suitable candidate datasets to be selected when testing a newly contrived time series forecasting method.

In the late 80's, under the influence of behaviourism and after the discovery of powerful gradient descent machine learning algorithms [44], researchers extensively tested artificial neural networks (such as multilayered perceptrons), feeding them with raw or preprocessed sequential data windows. However, although it was successfully proved that the artificial neural networks possess superior built-in curve fitting capability, unpreprocessed raw data feeding caused poor forecasting performance, mainly because forecasting time series is a global activity [40], while fitting is only a local one.

With the blooming of efficient clustering methods and the simultaneous advances in signal processing theory, it was noted that the forecasting performance of predictor systems could be improved if they could be trained on a sub population featured by common statistical properties; the research directions were then oriented to preprocess the raw datasets, before feeding the predictor systems.

One direction of research saw the analysis, detection and clustering of recurrent patterns detected in the raw series, in order to create homogeneous, representative training subsets, on which a dedicated machine could learn. Alternatively, researchers applied the signal processing theory to analyze the input series, isolate basic components, and finally relying on machines to the purpose of forecasting.

One of the first attempts to forecast time series using the pattern detection approach was proposed by Wasserman [49], who described simple yet efficient systems that train Ko-

honen Layers in order to find causal relationships between related patterns, and use such relations to perform predictions. Interesting approaches based on the use of artificial neural networks can be found in [16, 42]. Their research was original among the quantitative methods of financial forecasting. One of the first proposals of hybrid predictor systems can be found in [23], where authors developed a three stage forecasting process, involving a filter based feature selection; the clustering of training samples, performed using a SOFM algorithm; and finally the forecasting step executed with Support Vector Regression. Such method is reported to outperform the single SVR approach when tested to forecast the one step ahead of financial series such as the stock market price index. In [22] Hsu et al. proposed an approach to perform stock price forecasting by integrating a Self Organizing Map and Support Vector Regression (SOM-SVR), predicting the n steps ahead of a given time series, achieving better results when compared to older monolithic forecasting approaches. In [25], authors described a hybrid forecasting system composed of Self Organizing Maps and Least Square Support Vector Machines (SOM-LSSVM), candidating such alternative system as a general purpose predictor for time series and reporting successful benchmark tests, outperforming the single LSSVM method. More recently, Gencay et al. [21], starting from some stylized facts of the financial time series (such as volatility clustering and fat tails distribution of price returns), discovered new properties of financial volatility across different time scales (asymmetric vertical dependence). Since then, the prediction of financial time series making use of multiscale approaches has been a very active area of research (Rua and Nunes [43]; Masih and Alzahrani [36]; Francis and Sangbae [19]; Jammazi [26]; Reboredo and Rivera-Castro [41]). By the side of the frequency content analysis approach, such researchers analyzed the efficacy of multiscale approaches, particularly using space-frequency types of analysis, motivated by recent findings about the powerful methods of wavelets, the latter being applied either alone or in conjunction with other prediction or decomposition models.

Although the present work is all centered on forecasting time series, we never limited the scope of the research to the price predictions, nor we made assumptions on the statistical properties of the testing datasets. This should help to design machine learning frameworks candidate to be applied to a variety of application fields. Yet, we are convinced about the fact that large datasets such as the foreign exchange rates can help researchers to develop and test efficient machine learning predictor systems. Also, the difficulties related to the price forecast activity are useful to discard unproductive research paths as soon as they show inconsistencies or excessively inaccurate results.

At the beginning of 2014, after a burst of the volatility that brought the Bitcoin to a price peak over 1000 US Dollars, we searched and found the availability of full Bitcoin-USD currency cross price datasets. They were made available to the public once the traders' community requested fine granularity spot rates for speculation purposes. We examined the datasets in order to evaluate their quality and decided to select them to test our systems, as well as other benchmark forecasting methods. At the same time we considered the prediction of criptocurrencies one of the less known and explored areas of research, and considering the emerging importance of the commercial value of the Bitcoin, as well as its market capitalization, forecasting criptocurrencies could benefit not only to trading and investment but also to the hedging capability of enterprises which decided to transform their businesses adopting such new alternative payment method.

This thesis is organized as follows: Chapter 2 illustrates the first prediction system designed and developed using a Self Organizing Layer and a group of cooperating neural networks. Chapter 3 describes an hybrid time series prediction system designed using the Fast

Wavelet Transform for streaming time series analysis. Chapter 4 introduces a recent reassignment technique called Synchrosqueezing, as well as its mathematical foundations, and the proposal of a novel forecasting approach based on the Synchrosqueezed Continuous Wavelet Transform and machine learning objects employed to prediction purposes. Chapter 5 describes some implementation aspects related to the development of the prediction systems herein presented. Chapter 6 portrays conclusions and further research initiatives.

Chapter 2

Time series multiscale pattern recognition and forecasting

We herein present a prediction system composed of a self organizing layer and a group of artificial neural networks, whose purpose is to perform streaming time series analysis and forecast. The clustering layer is able to recognize incomplete subpatterns, populated with data sampled at different scales, all of them pointing to the incoming and unknown data element. Whenever a set of patterns is recognized, a raw forecast of the incoming value is performed using the information contained in the self organizing layer's selected prototypes. The group of artificial neural networks are trained to estimate the error measured between the raw forecast and the actual incoming value. Such estimate is used to fine tune the final real value forecast. Testing the system on a large dataset (the Bitcoin-US Dollar hourly exchange rates) we obtain good prediction results, outperforming the benchmark models. Finally we develop a simple trading strategy that uses the system predictions to perform spot market operations. The backtesting shows that such expert advisor can yield a 50% capital gain and a good profit factor.

Recently an alternative currency and payment method, named Bitcoin by its anonymous inventor and based on a public chain of digital signatures (Nakamoto, [39]), is becoming more and more popular, and could progressively become the competitor of other types of electronic payment. Although the system was introduced in 2009, its actual use began to kick in from 2013. Since then, the cryptocurrency's price, quoted against the principal fiat currencies, has suffered of high volatility and wide changes in value. Given the associated risk that both customers and vendors adopting Bitcoin were subject to, the rapid diffusion of the cryptocurrency was ultimately delayed. Nevertheless, Kondor et al. [33] showed that the Bitcoin network is exhibiting an exponential growth.

Given the potential impact of Bitcoin on both economy and finance, we focus on developing a system capable of producing accurate price forecasts of the new cryptocurrency.

Considerable research efforts have been dedicated in the past to the attempt to forecast the foreign exchange rates, see for example [16, 42]. Yet not much literature can be found on the forecast of Bitcoin indicators, the main reason lying in the relative shortness of price dataset series available for statistical analysis, in most cases limited to daily close price history.

A recent exploration path was followed by Shah and Zhang [45], who applied the Bayesian

Regression for Latent Source Model either for binary classification purposes or to estimate real valued variables, achieving good results with Bitcoin trading and at the same time giving scientific confirmation of the existence of price patterns in the mentioned time series.

More recently, other researchers [20, 28, 38] found evidence of correlation between the Bitcoin trading volumes and web search engines trends. Such works, however, are not directly aimed to perform a short term financial prediction of Bitcoin exchange rates and, as such, the possibility to develop a trading system, based on the cryptocurrencies' price forecast, remains only partially explored.

The existence of convincing literature focused on the analysis of Bitcoin was key to our decision to develop a different and novel approach, designed to perform short-term price predictions, and to test its accuracy against a benchmark model. Also, not many papers dealing with such time series using short sliding windows can be found, the main reason probably being the need for huge computational resources, proportional to the required precision of the pattern recognition operations.

Generally speaking, a system designed to perform a forecast of the series n steps ahead must be able to run in real-time configuration (data streaming from an external source); this can be achieved in more than one way.

One of the first attempts to forecast time series patterns was proposed in [49], in which a Kohonen Layer is trained with two related patterns simultaneously, thus being able to forecast the second one when only the first is given. More recently, in [22] Hsu *et alt.* proposed an approach to perform stock price forecasting by integrating a self-organizing map and support vector regression, predicting the n steps ahead of a given time series, achieving better results when compared to older monolithic forecasting approaches.

Starting from some stylized facts of the financial time series (such as volatility clustering and fat tails distribution of price returns), recently Gencay *et alt.* discovered new properties of financial volatility across different time scales (asymmetric vertical dependence) [21]. Since then, the prediction of financial time series making use of multiscale approaches has been a very active area of research ([19, 36, 41, 43]).

The herein proposed approach (far less explored in the literature) is to perform recognition using patterns composed only of the visible streamed data (incomplete patterns), considering the incoming (and missing) n values as those to be forecast.

As a direct consequence of this choice, the average precision of the pattern recognition operations is inevitably lower than the one obtained using full, complete patterns, since less information is available to test an input series against a database of prototypes. However, supposing that the clustering training is accurately performed, the gain in forecasting accuracy can be greater than those of the aforementioned approaches. In fact, if the pattern recognition error is contained within a range (hereafter referred to as "granularity parameter"), the one step ahead forecast is easy, since the last weight of the best matching prototype contains enough information to calculate a future value. On the contrary, if the clustering error is greater than the granularity, the best matching prototype does not give useful information, and a prediction cannot be correctly performed. For this reason, a multiscale approach is here considered and implemented: patterns composed of values sampled at different scales, derived from the original series, are concurrently used to overcome the challenges introduced by any inexact single recognition operation.

Selecting patterns having different scales is also consistent with the meaningful proposition of Ramsey [40] in that regression (fitting) a series is a local activity, whereas forecasting

is global in structure; this implies that an effective forecast cannot be performed unless sufficient data is appropriately selected from the time series and submitted to the predictor.

After having clustered the aforementioned price patterns, a fine tuning method (implemented using a set of cooperating multilayer perceptrons) is eventually used to achieve the final Bitcoin price forecast.

2.1 Method

Our system is composed of a self-organizing clustering layer and a group of cooperating artificial neural networks. The former was designed and implemented starting from the definition of the Kohonen Layers [30, 32], and adapted to the clustering difficulties associated to the cryptocurrency series' price and volatility. The chosen type of artificial neural network is the back propagation [44] multilayer perceptron (MLP), known to possess a good built-in capability to perform short-term forecasts of series characterized by some level of autocorrelation.

The clustering module is trained to cluster patterns derived from the Bitcoin price hourly series. Both the input patterns' generation procedure and the self-organizing layer training are explained next, subsequently the layers' real-time configuration is described. The purpose of the clustering layer is to be able to effectively recognize and select the clustering unit that best matches a price input pattern.

The recognition is performed, at each time step, for several price patterns at different scales. The prototypes that are found this way contain information useful to calculate a raw forecast of the Bitcoin price at the next time step.

A supplementary module is deputed to evaluate the forecasting error (against the incoming real price), and to create and maintain a preset number of moving averages of different period of such error history. Each moving average serves as input to its respective MLP, trained to estimate the error's one step ahead. The final forecast price of the Bitcoin is obtained by correcting the raw forecast with the average of the predicted errors.

Ideally, both the clustering layer and the cooperating neural networks should be retrained during the real time phase, in order to retain the stability of the forecasting performance level.

2.1.1 Self-organizing layers

Generally speaking, clustering a pattern means measuring the distance of such a pattern from each of the discovered prototypes, and selecting the best matching unit. The self-organizing layer is designed starting from the basic idea that the time series can be viewed as a series of concatenated patterns of fixed length, whose shape is unknown at the beginning, and must be explicitly found. To this purpose, we need a layer, initially empty, capable to expand whenever an unknown pattern is discovered.

Self Organizing Layers training

Patterns are tested against the prototypes and a best matching unit is evaluated. If the recognition error (euclidean distance) overcomes a granularity parameter (herein denoted Γ), the

layer grows with a new unit, whose training is immediately performed. Once the neuron's training is complete, it is encapsulated in the self organizing layer.

If the recognition error is less than the granularity instead, the layer can already cluster the testing pattern, and no training is required. In order to complete such unsupervised training phase, all the training data subset must be submitted to the layer.

Scaled patterns are extracted from the training dataset as follows: an iterator scans the data series (herein the Bitcoin hourly price close) BTCUSD H1 = $\{x_t : x \in \mathbb{R}^+, t \in \mathbb{Z}\}$, pointing at the current (last) timestep t_0. Denoting $K \in \mathbb{N}$ the input pattern size, and $\boldsymbol{S} = \{s_i : 1 \leqslant i \leqslant N\}$ a vector of $N \in \mathbb{N}$ scale coefficients, an intermediate matrix $\boldsymbol{M}$ of subpatterns $\boldsymbol{X}_i \in \mathbb{R}^K$ is generated:

$$\boldsymbol{S} = \left\{ s_1, s_2, ..., s_N \right\} \tag{2.1}$$

$$\boldsymbol{X}_i = \left\{ x_{t_0-(K-1)s_i}, \ ..., \ x_{t_0-2s_i}, \ x_{t_0-s_i}, \ x_{t_0} \right\} \tag{2.2}$$

$$\boldsymbol{M} = \left\{ \boldsymbol{X}_1, \boldsymbol{X}_2, ..., \boldsymbol{X}_N \right\} \tag{2.3}$$

Each of the $\boldsymbol{X}_i$ vectors constitute a subpattern having x_{t_0} as its last element (zero shift), its previous elements being obtained by back-sampling the series at constant intervals s_i. To be independent from the actual price value, at each time step each vector $\boldsymbol{X}_i \in \boldsymbol{M}$ is normalized to $\boldsymbol{X}'_i$, before being fed to the self-organizing layer:

$$l_2 \triangleq \left\| \boldsymbol{X}_i \right\|_2 \tag{2.4}$$

$$\boldsymbol{X}'_i = \frac{\boldsymbol{X}_i}{l_2} \tag{2.5}$$

It is worth noting that we use only one clustering layer, that is trained on all patterns sampled at different scales $s_i \in \boldsymbol{S}$. This avoids unit redundancy, preserves memory usage and favors the overall system's efficiency. Given a generic vector $\boldsymbol{v}$ of size $K \in \mathbb{N}$, let us define $\left\| \boldsymbol{v} \right\|_2^{\mathrm{k}}$ its L_2 norm calculated using just the vector's first $k \leqslant K$ elements:

$$\left\| \boldsymbol{v} \right\|_2^{\mathrm{k}} \triangleq \sqrt{\sum_{i=1}^{\mathrm{k}} {v_i}^2} \tag{2.6}$$

Let us call $J \in \mathbb{N}$ the size of the layer (the number of prototypes). The best matching unit's index $j_0, 1 \leqslant j_0 \leqslant J$, can be found with the following equations:

$$d_{i,j} = \left\| \boldsymbol{X}'_i - \boldsymbol{W}_j \right\|_2^{\mathrm{K}} \tag{2.7}$$

$$j_0 = \arg_j \min d_{i,j} \tag{2.8}$$

Eq.(2.7) means measuring the euclidean distance of the input pattern $\boldsymbol{X}'_i$ to each existing j^{th} prototype with weights $\boldsymbol{W}_j$, and selecting the closest unit (eq.(2.8)), having index j_0.

Whenever such clustering unit does not match the input pattern within the given granularity ($d_{i,j_0} > \Gamma$), a new clustering unit $\boldsymbol{W}'_{j_\varsigma}$ is created (see subprocess *SOL training* in fig. 2.1). $\boldsymbol{W}'_{j_\varsigma}$ weights are set perturbing those of $\boldsymbol{X}'_i$ by values drawn from a uniform random distribution with interval $\left[{}^{-\Gamma}/_{\sqrt{\mathrm{K}}}, {}^{\Gamma}/_{\sqrt{\mathrm{K}}}\right]$:

$$\begin{aligned} \boldsymbol{W}'_{j_\varsigma} &= \boldsymbol{X}'_i + r \\ r &= unif\left(-\frac{\Gamma}{\sqrt{\mathrm{K}}}, \frac{\Gamma}{\sqrt{\mathrm{K}}}\right) \end{aligned} \tag{2.9}$$

The weights are perturbed in order to improve the generalization power of the new clustering unit. With respect to Self Organizing Maps [30–32], our network does not adjust the weights when a new input matches a clustering unit, and no neighborhood parameter exists. Similarities to the Latent Source Model (Chen *et al.*) [6] arise as the granularity Γ is progressively set to a value near zero, except that our system never tries explicit pattern classification.

Self Organizing Layers testing

In a real-time configuration, all the elements of a fixed size pattern are visible except for those to be forecast. Let us denote τ as the number of visible elements of an input pattern $\boldsymbol{X}_i$. Recognizing a normalized input pattern $\boldsymbol{X}'_i$ (hence using only the first $\tau < K$ elements), is equivalent to selecting a best matching unit as follows:

$$\epsilon_{i,j} = \left\| \boldsymbol{X}'_i - \boldsymbol{W_j} \right\|_2^\tau \tag{2.10}$$

$$\epsilon_{i,j} \leqslant d_{i,j} \leqslant \Gamma \tag{2.11}$$

$$j_0 = \arg_j \min \epsilon_i \tag{2.12}$$

The best matching prototypes whose recognition error $\epsilon_{i,j_0} > \Gamma$ are discarded (see subprocess *SOL testing* in fig. 2.1). We denote η as the number of prototypes that match the test input correctly; in other words, $\eta \leqslant N$ are the remaining prototypes satisfying eq. (2.11). They are selected to extract a vector $\boldsymbol{w}_K$ of η useful weights. Let us consider also a vector $\boldsymbol{l}_2$ holding the η normalization factors of the $\boldsymbol{X}_i \in \boldsymbol{M}$ input vectors correctly recognized by the system:

$$\boldsymbol{w}_K = \left\{ w_{1,K}, w_{2,K}, ..., w_{\eta,K} \right\} \tag{2.13}$$

$$\boldsymbol{l}_2 = \left\{ l_{2(\boldsymbol{X}_1)}, l_{2(\boldsymbol{X}_2)}, ..., l_{2(\boldsymbol{X}_\eta)} \right\} \tag{2.14}$$

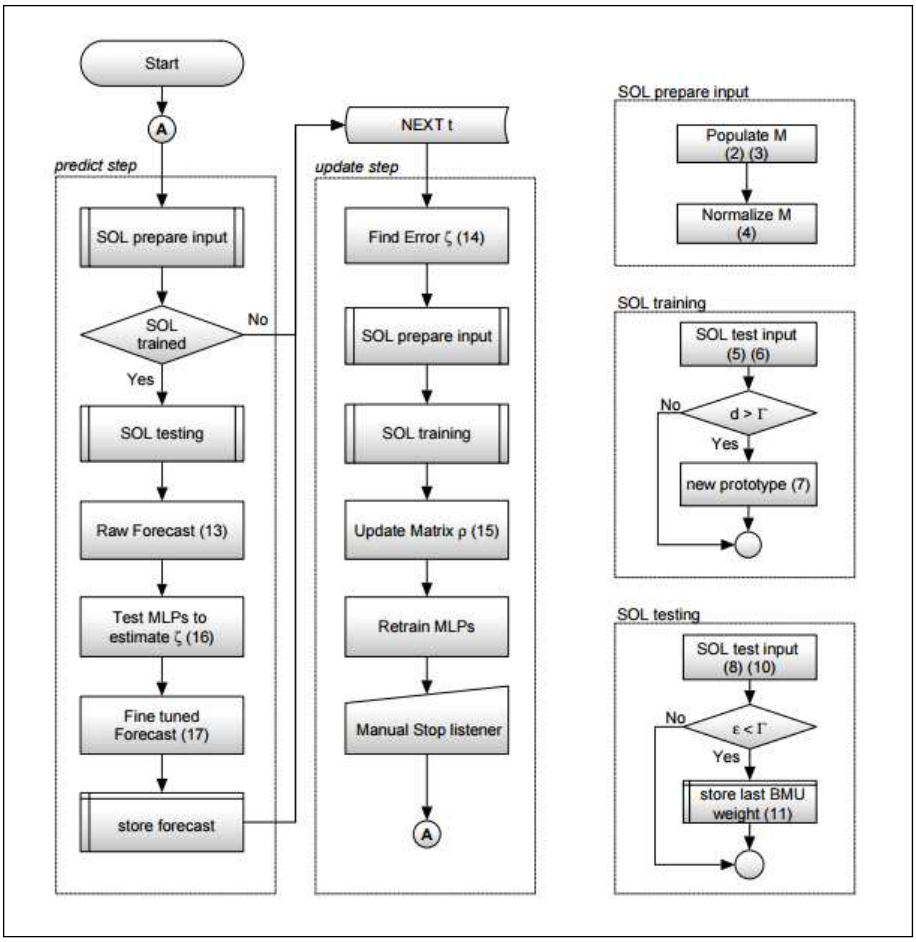

Figure 2.1: System flow chart. Left column: operations of the predict step; centre column: update step; right colums: subprocesses. Predict and Update steps are sequential. During the predict step, the forecast is performed only if the Self-organizing Layer has been fully trained over the entire training dataset partition (decision box *SOL trained*). The update step is performed when a new external data sample has entered the system. During the update step, both SOL and MLPs are retrained. Finally, a new cycle begins whenever the control flows through connector *A*. The *SOL prepare input* subprocess gathers the operations needed to create the matrix of multiscaled patterns and to normalize the data. The *SOL training* subprocess box shows the applications of eqs.(2.7), (2.9). The *SOL testing* subprocess box shows how only the best-matching prototypes, whose recognition error $\epsilon < \Gamma$, contribute to the raw forecast formation.

Let us denote $\left\langle \boldsymbol{v}_1 \, \boldsymbol{v}_2 \right\rangle \triangleq \boldsymbol{v}_1^{\mathrm{T}} \boldsymbol{v}_2$ as the scalar product of two generic vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$. Supposing $\tau = K-1$, (the system is set to forecast the series one step ahead) a raw forecast $\hat{u}_{t_0}$ of the real value x_{t_0} can be calculated by averaging the denormalized weights $w_{i,K} \in \boldsymbol{w}_K$:

$$\hat{u}_{t_0} = \eta^{-1} \left\langle \boldsymbol{l}_2 \, \boldsymbol{w}_K \right\rangle \tag{2.15}$$

Uncertainty introduced by the lack of information regarding the unknown last pattern elements induces a forecasting error, denoted as ζ_t:

$$\zeta_t = x_t - \hat{u}_t \tag{2.16}$$

As shown in fig. 2.1, ζ_t can be calculated as soon as the system receives a new incoming series value. If we were able to correctly estimate ζ_t, it would be possible to use such value to fine tune the raw forecast $\hat{u}_{t_0}$, hence obtaining a much more accurate price forecast. This is the purpose of a group of cooperating back propagation multilayer perceptrons, each trained to forecast a moving average of past history errors ζ_t, as described in the next subsection.

2.1.2 Cooperating back propagation multilayer perceptrons

Multilayer perceptrons are used as a mean to predict a correction to the raw forecast performed with eq. (2.15), in order to fine tune the real value price forecast $\hat{x}_t$. MLPs are trained using the last, small portion of the training dataset, before the system is put to work. The raw forecast error ζ_t is stored at each time step. Such sequence of error values is used to calculate a set of N moving averages of different periods p_i, $1 \leqslant i \leqslant N$, the last H elements of which are input queues $\boldsymbol{\rho}_{p_i}$ to their respective MLP, as columns of the following matrix $\boldsymbol{\varrho} \in \mathbb{R}^{H \times N}$:

$$\boldsymbol{\varrho}_{H,N} = \begin{pmatrix} \rho_{1,p_1} & \rho_{1,p_2} & \cdots & \rho_{1,p_N} \\ \rho_{2,p_1} & \rho_{2,p_2} & \cdots & \rho_{2,p_N} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{H,p_1} & \rho_{H,p_2} & \cdots & \rho_{H,p_N} \end{pmatrix} \tag{2.17}$$

The use of smoothed sequences (such as the above-mentioned moving averages) as input to the MLPs improves their overall prediction accuracy.

During the training phase, the incoming series ζ_t is used to calculate N target values ρ_{H+1,p_i}, necessary to train each respective i^{th} MLP to produce an output $\hat{\rho}_{p_i} \approx \rho_{H+1,p_i}$.

In the real time phase, $\hat{\rho}_{p_i}$ is the output of each i^{th} neural network. Such N forecast values allow each estimated error $\hat{\zeta}_{t_i}$ to be found. The final error estimate $\hat{\zeta}_t$ is calculated averaging all the $\hat{\zeta}_{t_i}$, and it is used to produce the final price forecast output:

$$\hat{\zeta}_t = N^{-1} \sum_{i}^{N} \left(p_i \, \hat{\rho}_{p_i} - \sum_{m=1}^{p_i - 1} \zeta_{t-m} \right) \tag{2.18}$$

$$\hat{x}_t = \hat{u}_t + \hat{\zeta}_t \tag{2.19}$$

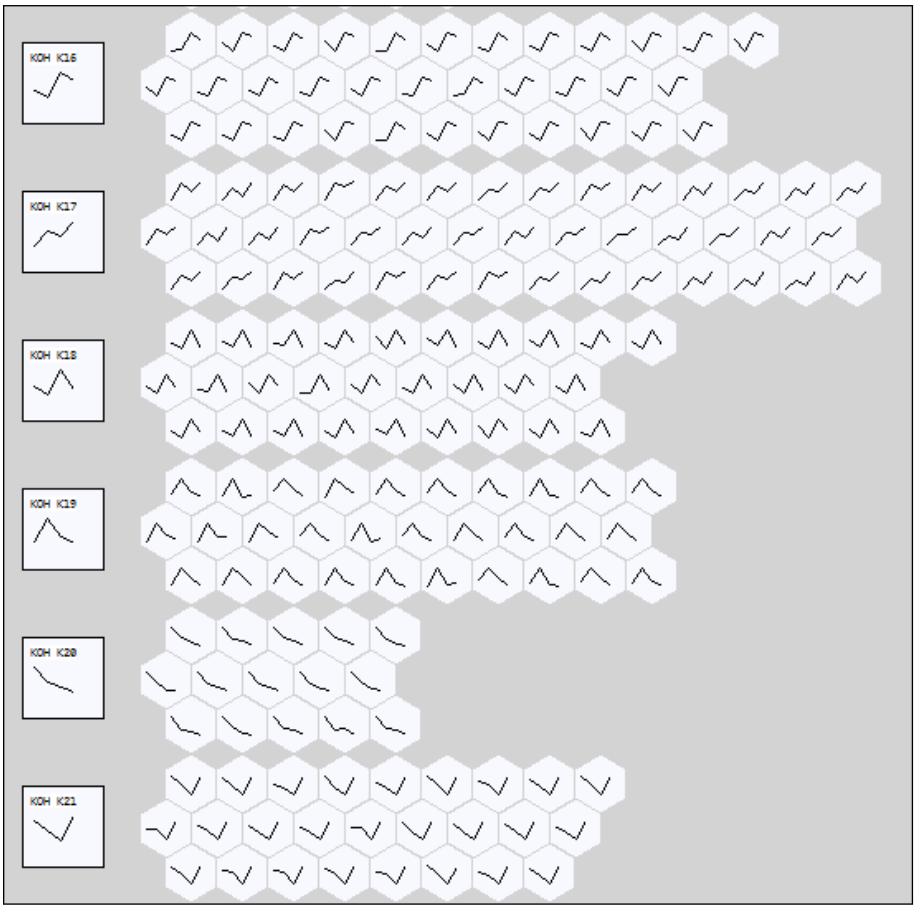

Figure 2.2: Bitcoin price patterns clustering operations. Left: a selection of some prototype patterns; right: a set of input patterns correspondingly recognized during the testing phase. The used granularity parameter is $\Gamma = 0.004$. The lower Γ, the higher the shape similarity among the clustered patterns and, consequently, the lower the number of patterns associated to the same cluster. In order to use the clusters for effective prediction purposes, they must contain only motifs exhibiting strong similarity. The adequate choice of Γ is key to obtain good system output performance.

Upon arrival of a new dataset value, the actual ζ_t is available and can be stored. Hence, it is possible to find $\rho_{H+1,p_i}, \forall i$, and use such values to retrain each respective i^{th} MLP, using the same procedure followed during the initial training phase.

2.2 Results and discussion

Observing the cryptocurrency's price series, (BTCUSD, presented as a weekly candlestick chart in Fig. 2.3), it can be noted that it shows an overall appreciation and volatility spark during the last quarter of 2013, the price reaching a peak on December 4^{th}, 2013. During 2014 instead, Bitcoin showed a well defined downtrending correction and slowly decaying volatility as the price tested and then lowered below the psychological support (then resistance) level of 300 USD.

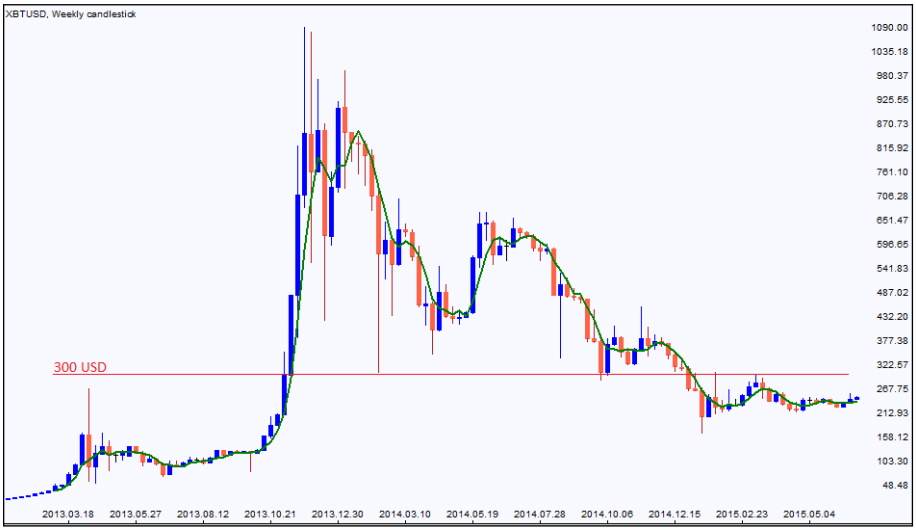

Figure 2.3: BTCUSD weekly candlestick chart, showing the Bitcoin price for the period January 2013 to June 2015. The solid line is a simple moving average of period 4. The horizontal light line is set at the psychological level of 300 USD. From the candlestick chart we can appreciate that this market is actually respecting such level (reached by the price numerous times).

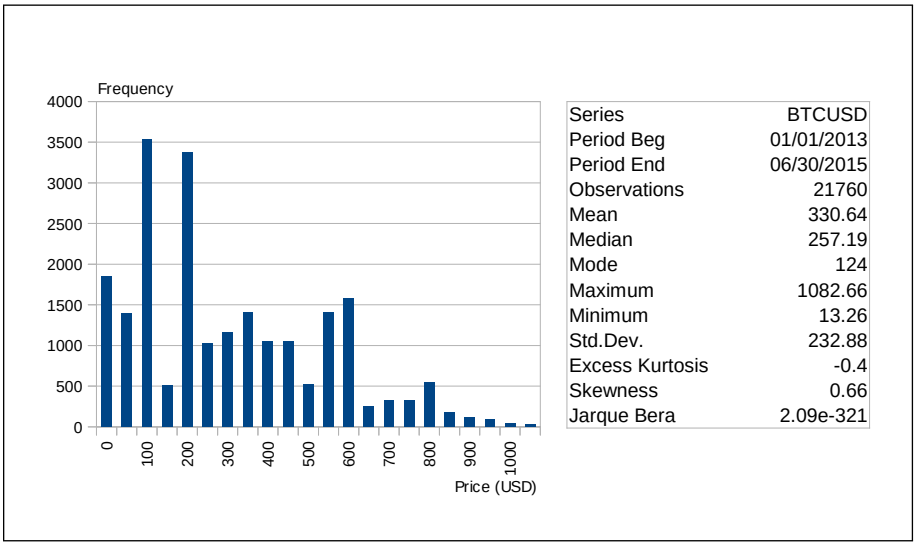

Figure 2.4: BTCUSD hourly close price distribution summary statistics for 21760 observations, spanning from 01/01/2013 to 06/30/2015. The dataset consists of more than 20 thousands hourly samples of the Bitcoin exchange rate. Note that the cryptocurrency average price (against USD), is just 30.64 USD above the meaningful psychological support level of 300 USD, (Fig. 2.3).

The dataset we used is the Bitcoin price hourly series, spanning from January 1st 2013 to June 30th 2015. Data inspection reveals an overall good quality of the whole set, since only 13 gaps can be found (103 missing bars, 21760/21863 observations, i.e. 0.47%). A statistical synthesis of the hourly data series is presented in Fig. 2.4, showing platykurtosis and a slight skewness; Jarque-Bera test confirms that the series is non normal. The whole dataset is partitioned into an in-sample period, used for training and validation of the system, (from January 1st 2013 to December 31st 2014), and an out-of-sample period (from January 1st 2015 to June 30th 2015) to test the system on real data.

After extensive tests, we decided to set the pattern size $K=8$, and the granularity $\Gamma = .004$. Consequently, the self organizing layer grows in order to cluster patterns of this same size whenever Γ is exceeded. Lower values of the granularity cause the number of discovered patterns to be close to its theoretical maximum (i.e. the total number of patterns, obtained sliding step by step the training series with a window of fixed size K, repeating the operation for each sampling rate, defined by the scale value s_i). As Γ tends to 0, the self organizing layer behaves similarly to a Latent Source Model container. On the contrary, higher values of the granularity cause the number of clusters (and the clustering precision) to reduce.

The value we found for Γ is small enough to avoid the trivial match effect, which happens when two adjacent patterns, shifted by one step, are clustered together. Fig. 2.2 shows some of the clustering units found, with the corresponding real time patterns found by the system.

Once the self organizing layer is trained, it must be able to perform recognition of patterns synthesized using a multiscale approach. During our tests, we found that the sequence of scale coefficients $\boldsymbol{S} = \{2^m : 0 \leqslant m \leqslant 7,\ m \in \mathbb{N}\}$ is enough to achieve a better forecasting ability than that of a naive predictor. Also, as described by eq.(2.2), the scaled input patterns always have the last element pointed to the forecasting step. Hence, when running, the self organizing layer is called to perform recognition of patterns composed of $\tau = K-1 = 7$ elements, as indicated by eqs. (2.10) and (2.12).

Raw forecast errors ζ_t, are stored in a queue whose size is sufficient to accommodate enough data to calculate all the moving average sequences of different period. Finally, for the scope of this research, the neural network configuration was set as follows: standard input layer size 8, triple hidden layer bank (sizes 8, 8, 16), output layer size 1; hyperbolic tangent activation function, full neural interconnection. We performed several tests with different network parameters, confirming our intuition that the number and size of hidden layers of the MLP should be kept low in this type of system. The reason lies in the fact that the purpose of the MLP is to forecast the next step value of a smoothed series.

We compared the price returns forecasts, performed for the whole out of sample period (4334 prediction attempts), with a naive benchmark method which, despite its simplicity, is generally recognized as hard to beat in terms of financial price forecasting accuracy when using exclusively technical analysis indicators, or when compared to quantitative finance models – see for instance [17]. Let us recall that naive methods are generally designed to forecast a time series using the most recent observed change trend. If the last occurred price change is taken into consideration (as we did in the present work), the naive model is expressed by the relation $\hat{x}_t = x_{t-1} + \Delta x_{t-1}$, where $\Delta x_{t-1} \triangleq x_{t-1} - x_{t-2}$. We also tested the raw output of the clustering system performed using the self organizing layers described in the sections above (see eq. (2.15)), without applying any error correction through the MLP. The results of a forecasting session is depicted in Fig. 2.5, showing Bitcoin price returns predicted for the hourly close one step ahead, during the out-of-sample period. Sorted absolute output errors are shown in Fig. 2.6. Occurrence of errors less than 50 cents is 28% for the naive,

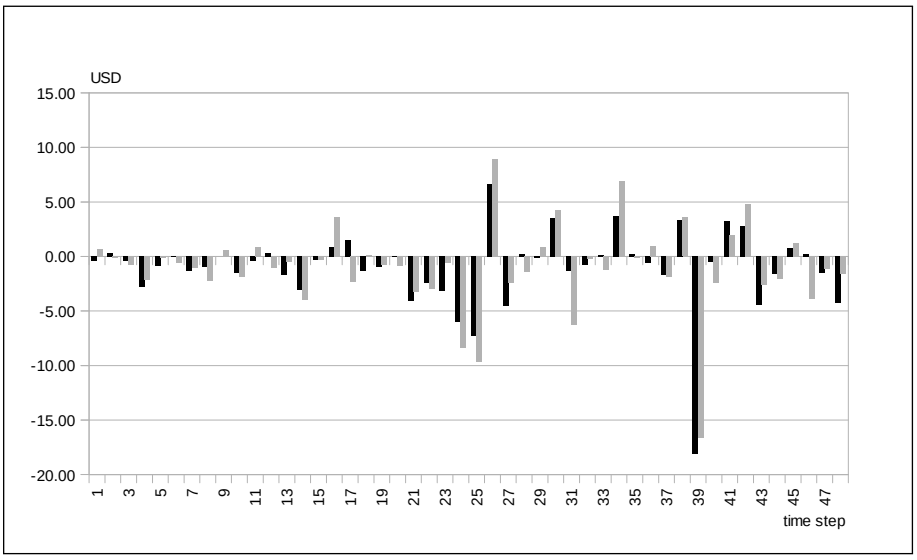

Figure 2.5: Hourly Bitcoin price forecast session, January 2015. Actual raw price returns in dark histogram columns, forecast price returns in light. Note that the accuracy of the forecast change of direction - CoD of the price returns is directly proportional to the absolute value of the respective returns. In other words, the system's prediction is more accurate, in terms of CoD, when the market is volatile.

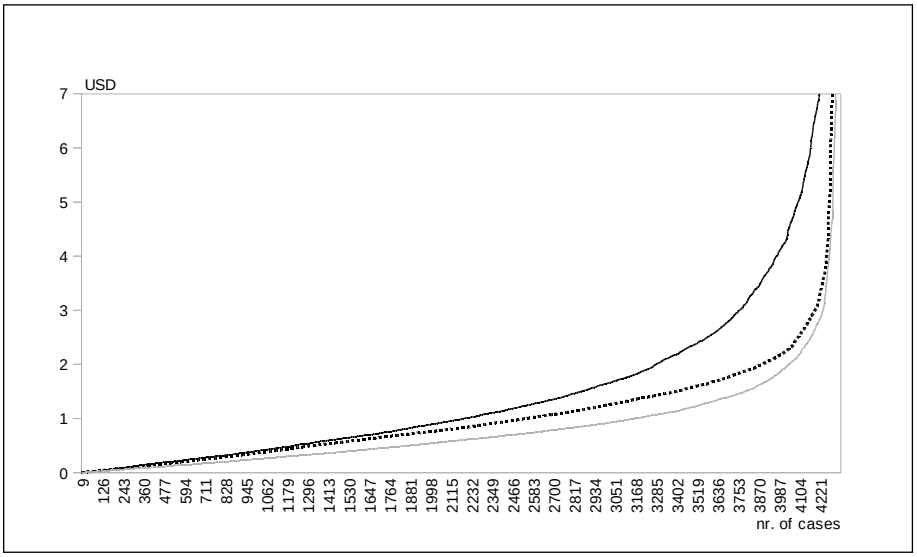

Figure 2.6: Sorted absolute errors of the price forecast session: naive forecast absolute errors (dark solid line), raw forecast with no MLP correction absolute errors (dotted line) and system absolute errors (light solid line). Occurrence of absolute errors less than 50 cents is 28% for the naive predictor, 30% performing a raw forecast, 43% for the system, indicating that the proposed approach achieves the highest forecasting accuracy.

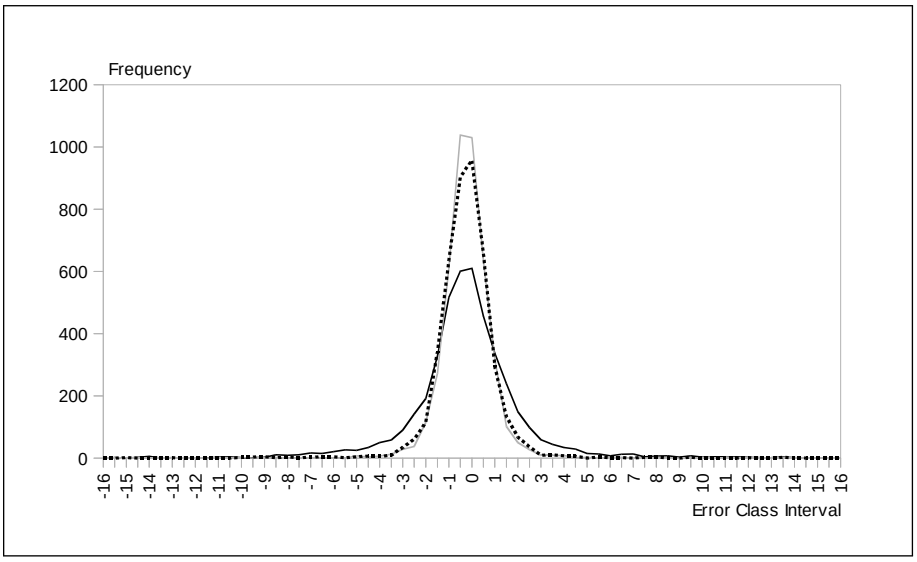

Figure 2.7: Raw error distribution of the price forecast for the whole out-of-sample session: naive forecast error (dark solid line), raw forecast error (dotted line), proposed system error (light solid line). System error shows higher kurtosis, confirming that its smaller error classes are more frequent if compared to the same classes of the benchmark models.

whereas for the raw forecast (performed without the intervention of the MLPs), it is 30%. The proposed complete system increases such statistic up to 43%. The naive method cumulative absolute error of the whole forecast session (4334 prediction attempts) is 7025.01 USD, whereas the raw forecast one is 4892.10 USD. The proposed system shows an error as low as 3662.22 USD. Moreover, in 90% of the times, the naive method and the raw forecast perform respectively with an error less than 3.64 and 2.03 USD, whereas the proposed system has an error as low as 1.67 USD. Error distributions for both the system and the benchmark methods are plotted in Fig. 2.7. The system error distribution denotes higher kurtosis and skewness than the same statistics of the benchmark methods, whereas, not surprisingly, the naive error distribution is characterized by fatter tails.

2.3 Profitability tests

We implemented a simple high frequency trading strategy based on the Bitcoin hourly close forecast (performed with the herein presented model), using no financial leverage and a reward to risk ratio set to the bare minimum of 1.2, by means of fixed automatic take-profit and stop-loss orders associated to each market operation. The trading strategy is developed as an expert advisor with real time operational capability. The forecasting phase is performed after each incoming price value, using the herein described hybrid prediction system. Having forecasted the one step ahead hourly price close, the expert evaluates market price at each tick, waiting for an opportunity to buy at lower price or to sell at higher price in the direction of the forecast price. An order is entered only if the projected return is greater than the absolute loss imposed with the automatic stop set for the specific operation. For the sake of simplicity, stop losses are set at a constant price difference from the entry point throughout the whole backtesting session. In order to improve the expert's profitability, one could contrive more sophisticated stop loss calculation methods, for example taking account of

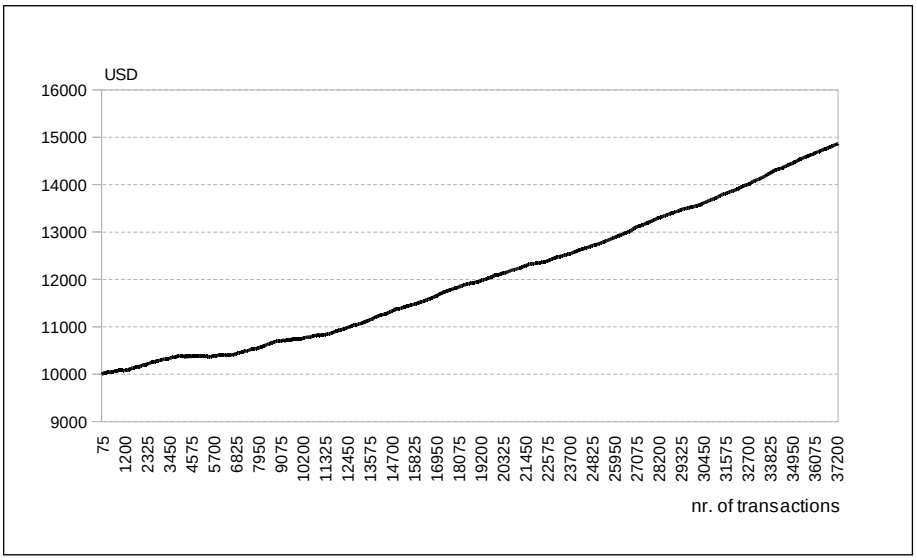

Figure 2.8: Cumulative Profit/Loss of a high frequency strategy, tested for the out-of-sample period (first semester, 2015), executing 37245 long and short operations. A 3.4 pips spread is applied upon order opening. The overall ROI is 48.6%, with a maximum drawdown of 0.31%.

the actual daily market volatility. We tested such strategy in the out-of-sample period. Results show a profit factor greater than 1.5, with successful operations accounting for the 50% of the total. Cumulative profit and loss graph is depicted in Fig. 2.8, describing a semi annual return on investment of 48.6% and a maximum drawdown of 0.31%. This performance takes into account transaction costs of 3.4 dollar cents for each operation (a simulated retail financial service provider's profit).

Finally, we repeated the tests on both British Pound - USD (GBPUSD) and Euro - USD (EURUSD) hourly exchange rates, for the period January 2014 - May 2016 (using the period up to November 2015 for training). With these data, the simulated semi annual profit gained on May 2016 is, respectively, 40.94% and 41.85%. These results confirm the applicability of our approach also for price series different from Bitcoin price.

2.4 Further developments

Given the potential wide impact of Bitcoin price and volatility, we proposed a model aimed to generate accurate price forecasts, and we evaluated its performance on the cryptocurrency's hourly close series. The system is composed of a self-organizing layer, a data processing module, and a group of cooperating back propagation MLPs. The forecasting process is achieved in two steps. First, a set of patterns of fixed length is generated by sampling the source data series at different scales, then recognized by the clustering layer, and a raw price forecast is performed using the information contained in the best matching prototypes. Second, each cooperating MLP performs a forecast of the error of the value produced by the first step; the average of such values is used to fine tune the real value forecast.

When tested on a large session of the Bitcoin - USD hourly exchange rates (21760 observations), the systems shows its ability to outperform the benchmark methods, the latter not using multiscale pattern recognition procedures. Such result confirms that a forecasting operation is global in nature, and it becomes feasible whenever a system, aimed to perform an

accurate prediction, operates simultaneously on the information retrieved from overlapping input ranges selected at constant intervals from the source series.

Despite possible initial lengthy training operations, especially regarding the population of the clustering self organizing layer, we believe that the system's performance could be greatly improved reducing the granularity parameter, thus creating larger databases of prototypes used to specialize each responsible predictor machine. Also further tests, aimed at improving the system's clustering capabilities after having substituted the self organizing layers euclidean distance kernel with different times series equality statistical measures, are already in place. This is the direction of further research initiatives, aimed to improve the forecasting reliability and accuracy of multiscale pattern recognition assisted predictor systems.

Chapter 3

Multiresolution analysis and prediction of streaming datasets

In this chapter we explore the shift variance of the fast wavelet transform - FWT, and we prove a novel theorem suitable to build efficient prediction systems of streaming univariate time series, making use of specialized statistical estimators or neural regressors trained to predict the one step ahead discrete wavelet transform. An effective real value prediction can then be obtained performing the inverse DWT of such estimated crystal, although several optimization steps can be taken to further improve the precision of the forecasting results. When tested on the Bitcoin hourly exchange rates, as well as on different statistical time series, the results are encouraging and fostering further research initiatives aimed at the improvement of the forecasting capability of prediction systems using discrete domain wavelet analysis of digital signals.

Generally speaking we believe that the prediction of streaming datasets will become a core issue with the advancement of technologies such as the Internet of Things, giving the emerging need for real time series forecasting of data generated by sensors and surveillance devices. The prediction of time series is also important for a wide range of application fields: in the present thesis we focus on the analysis and prediction of the currency exchange rates (the Bitcoin - USD currency pair in the specific), since it is possible to retrieve large chunks of the aforementioned financial series, and such opportunity greatly fosters the development and testing of any non-parametric contrivable prediction system.

The problem of forecasting streaming datasets has been largely explored in the past, and recently researchers [26, 36] are heading towards multiresolution approaches, particularly using space-frequency types of analysis, motivated by recent findings about the powerful methods of wavelets, the latter being applied either alone or in conjunction with other prediction or decomposition models.

Recently Daubechies *et alt.* [12] provided the necessary theoretical guarantees to the empirical results obtained by Huang [24] with the proposal of the Empirical Mode Decomposition and its successive improvements [50] (Wu and Huang), which allowed the scientific community to approach the time series analysis with a newer, yet simple, different view than the classical Fourier analysis.

At the same time, the explosion of wavelet approaches to general time series analysis seems to converge to the application of un-decimated discrete wavelet transform - DWT

methods (also Stationary, or "à trous", or redundant DWT), in a wide variety of scientific disciplines; the reason being for shift invariant applicability, though resource expensive, of the aforesaid methods to the analysis of streaming datasets.

We are interested in exploring the actual need for redundancy in applying the DWT to real value forecasting purposes; and dealing with the shift variance problem, particularly referenced to streaming input datasets, with the aim to build a general framework suited to be effectively used for time series analysis and prediction.

We prove a novel theorem on the shift variance of the convolutional, decimated DWT (also known as fast wavelet transform) of univariate sampled series, and we apply it in order to improve a prediction's system accuracy, also using some previous experience on the development of artificial neural networks and self organizing clustering machines, as outlined in the previous chapter of this thesis. Results show that the suggested method can outperform the benchmark models when applied to the testing financial series, as well as to other statistical data sets.

This chapter is organized as follows: section 3.1 provides a description of the proposed approach; the section is further subdivided into several subsections, each of which illustrates the theoretical guarantees and the specifics of the proposed prediction system. Section 3.2 provides a discussion of the research results. Section 3.3, portrays a direction to further research activities.

3.1 Method

In order to give a brief review of the discrete wavelet transform's theory and, contextually, to introduce the notation, let us recall that the families of wavelets:

$$h_{m,n}(t) = \left\{ a_0^{-m/2}\, h(a_0^{-m} t - n b_0) : m, n \in \mathbb{Z}; a_0, b_0 \in \mathbb{Z}_0 \right\} \tag{3.1}$$

have been particularly applied, in the past, to the analysis of signals pertaining to many scientific disciplines, and recently they were also applied to the study of financial time series. The family $h_{m,n}$ constitutes a set of different shifted and scaled versions of a commonly applied mother wavelet, which literature also generally denotes $\psi(t)$. The naming "discrete" wavelet transform derives from choosing the parameters a_0 and b_0 from a discrete sublattice. If the dilation parameter a_0 and the translation parameter b_0 are respectively chosen very close to 1 and 0, the resulting frame $\{h_{m,n} : m, n \in \mathbb{Z}\}$ is very redundant and close to a continuous family of wavelets [10]. If those parameters are set as $a_0 = 2$ and $b_0 = 1$, redundancy is avoided, although the $h_{m,n}$ functions constitute an orthonormal basis of the vector space of measurable, square integrable 1D functions $L^2(\mathbb{R})$ and they become:

$$h_{m,n}(t) = \left\{ 2^{-m/2}\, h(2^{-m} t - n) : m, n \in \mathbb{Z} \right\} \tag{3.2}$$

Their elements have good localization properties in both the spatial and Fourier domains. If the source time series variable is continuous, its DWT is then defined as:

$$T_{m,n}(f) = \left\langle h_{m,n}, f \right\rangle = 2^{-m/2} \int_{\mathbb{R}} f(t)\, h(2^{-m} t - n)\, \mathrm{d}t \tag{3.3}$$

Mallat proved that functions $f(t) \in L^2(\mathbb{R})$ can be considered as a limit of successive approximations (smoothed versions of $f(t)$), and that it is possible to find the wavelet coefficients $T_{m,n}$ as the difference of two approximations of $f(t)$ at consecutive different scales [34]. To obtain this, it is necessary to define two families of scaling and wavelet functions $\phi_{m_0,n}(t)$ and $\psi_{m,n}(t)$, with which it is possible to express $f(t)$ by means of the following *wavelet series*:

$$f(t) = \sum_n c_{m_0,n}\phi_{m_0,n}(t) + \sum_m \sum_n d_{m,n}\psi_{m,n}(t) \tag{3.4}$$

Eq.(3.4) is the key to signal reconstruction (also inverse DWT - IDWT). The forward DWT serves to retrieve the $c_{m_0,n}$ and $d_{m,n}$ sets of coefficients:

$$\begin{cases} c_{m_0,n}(f) = \displaystyle\int_{\mathbb{R}} f(t)\,\phi_{m_0,n}(t)\,\mathrm{d}t \\ \\ d_{m,n}(f) = \displaystyle\int_{\mathbb{R}} f(t)\,\psi_{m,n}(t)\,\mathrm{d}t \end{cases} \tag{3.5}$$

In practice, however, our objective is to work with a sampled series $\boldsymbol{X}$ of size $n_{\boldsymbol{X}}$, composed of the last consecutive sampled values of $f(t)$,

$$\boldsymbol{X} = \left\{ x_{n_{\boldsymbol{X}}-1}, ..., x_1, x_0 \right\} \tag{3.6}$$

In eq.(3.6) the x_0 element denotes the newly inserted element. Considering a streaming dataset, the size $n_{\boldsymbol{X}}$ is kept constant by popping (removing) the first and elder element from vector $\boldsymbol{X}$ (*first in last out*). In order to be able to work with sampled series $\boldsymbol{X}$, equations (3.4) and (3.5) must be discretized: $\phi(t)$ and $\psi(t)$ values are calculated, while integrals are replaced by summations.

However, a multiresolution signal decomposition of a sampled series can be performed efficiently in a recursive way, by means of filtering and downsampling operations. Normally those filters are denoted $\boldsymbol{h}$ and $\boldsymbol{g}$ and their size is herein denoted as $n_{\boldsymbol{h}}$. Because of their properties, in signal analysis they are usually referred to as quadrature mirror filters. Descriptions of the FWT algorithm to which the interested reader can refer is found in [4, 9] while a dissertation on the relationship between the Wavelet Series and the Discrete Wavelet Transform can be found in [51]. Let us denote $n_{\boldsymbol{h}}$ the size of both filters, m as the recursion depth of the procedure, 2^{-m} the resolution $\mathscr{R}_m$ to which the signal is analyzed at depth m. If $m = 0$ ($\mathscr{R}_0 = 2^0 = 1$), the input series is of course the source series itself. The forward DWT starts by convolving the source series with both filters $\boldsymbol{h}$ and $\boldsymbol{g}$, and retaining (separately) one sample out of two. This allows to obtain respectively two sets of coefficients, herein denoted by c_1 and d_1, of the next level $m = 1$ (lower resolution $\mathscr{R}_1 = 2^{-1}$), respectively representing a smoothed version of $\boldsymbol{X}$ and the difference of information between the two adjacent series $c_0 \triangleq \boldsymbol{X}$ and c_1. The resulting vector of c_1 coefficients serves as input to the operation of the next level $m = 2$, and the process is repeated until the maximum depth M is reached. The last set of coefficients c_M is retained. The maximum depth M depends on both $n_{\boldsymbol{X}}$ and $n_{\boldsymbol{h}}$:

$$M = log_2 n_{\boldsymbol{X}} - \lceil log_2 n_{\boldsymbol{h}} \rceil \tag{3.7}$$

The first term of eq. (3.7) takes the input source size into account, expressing the maximum number of downsampling operations that can be performed. The second term takes the filter size into account, since a convolution cannot be performed if the source input size is shorter than the size of the filter itself. (At the same time, a source input must always be a sequence whose size is a power of 2, hence the need to approximate at ceiling the value $log_2 n_{\boldsymbol{h}}$). As a consequence of eq.(3.7), performing a forward DWT of a source input $\boldsymbol{X}$ to the maximum depth M constrains $n_{\boldsymbol{c}_M}$ to be:

$$n_{\boldsymbol{c}_M} = \frac{n_{\boldsymbol{X}}}{2^M} \tag{3.8}$$

and $n_{\boldsymbol{d}_m}$ (the size of the vector containing the difference of information between two adjacent c_{m-1}, c_m series) to be:

$$n_{\boldsymbol{d}_m} = \frac{n_{\boldsymbol{X}}}{2^m} \tag{3.9}$$

At the recursion end (completion of the operation), the resulting vector $\boldsymbol{T}$ of forward transform coefficients of the series $\boldsymbol{X}$ is populated in the following order:

$$\begin{aligned} \boldsymbol{T} = \Big\{ & c_{M,0}, c_{M,1}, ..., c_{M,n_{\boldsymbol{c}_M}}, \\ & d_{M,0}, d_{M,1}, ..., d_{M,n_{\boldsymbol{c}_M}}, \\ & d_{M-1,0}, d_{M-1,1}, ..., d_{M-1,n_{\boldsymbol{d}_{M-1}}}, \\ & d_{M-2,0}, d_{M-2,1}, ..., d_{M-2,n_{\boldsymbol{d}_{M-2}}}, \\ & ... \\ & d_{1,0}, d_{1,1}, ..., d_{1,n_{\boldsymbol{d}_1}} \Big\} \end{aligned} \tag{3.10}$$

Note that the size of $\boldsymbol{T}$ equals the size of the source series $n_{\boldsymbol{X}}$. Hereafter, the subsets of coefficients allocated at the same recursion depth m are also referred to as *subbands* B_m.

The aforesaid recursive forward FWT procedure suffers of a principal drawback: in case of streaming datasets, the computation on subsequent sampled windows is affected by a *shift variance* problem, since at each insertion of a new element in the source series, many DWT coefficients (located in different subbands) are affected. As a direct consequence of this problem, different shift invariant DWT algorithms were found (they are referred to as Maximum Overlap, "a trous", stationary, or undecimated DWT). All of them are expensive in terms of memory and computational resources needed (high number of containers holding portions of the MRA coefficients). Also, if multiple undecimated MRA series must be held (eg. for machine learning purposes), this implies working with tensors of coefficients.

Instead note that, considering a single series of size $n_{\boldsymbol{X}}$, and wavelet filters of size $n_{\boldsymbol{h}}$, its multiresolution analysis performed using the decimated DWT implies a number of operations $O = 2n_{\boldsymbol{X}}(2n_{\boldsymbol{h}} - 1)$, as also outlined later (see *th.* on the computational efficiency), hence the algorithm's efficiency is comparable to the one of the Fast Fourier Transform. More over, since no splitting operations are performed on the source series, a CPU implementation of such algorithm was tested nearly as efficient as the one factoring wavelet transforms into lifting steps, despite the latter's theoretical higher efficiency [13].

Having a digital source signal, sampled at constant time intervals, and performing a multiresolution analysis -MRA on a pattern of fixed size $n_{\boldsymbol{X}}$, a matrix $\boldsymbol{Q}_{n_T, n_{\boldsymbol{X}}}$, can be filled with

		N								
		2	3	4	5	6	7	8	9	10
m	1	2	3	4	5	6	7	8	9	10
	2	3	4	6	7	9	10	12	13	15
	3	3	5	7	8	10	12	14	15	17
	4	3	5	7	9	11	13	15	16	18
	5	3	5	7	9	11	13	15	17	19
	6	3	5	7	9	11	13	15	17	19
	7	...								

Table 3.1: Number of varying coefficients at each depth m in a decimated DWT, performed using Daubechies orthogonal wavelet filters (filter size $n_{\boldsymbol{H}} = 2N$) on a streaming source series. The maximum reachable recursion depth m depends also on the size of the input series, which is not considered here. From recursion depth m onward, the convergence of varying quantities is stable.

$n_{\boldsymbol{T}}$ MRA row entries, calculated at each shift-insert operation to account for an incoming dataset element x_0 pushed into the source vector $\boldsymbol{X}$:

$$\boldsymbol{Q}_{n_{\boldsymbol{T}},n_{\boldsymbol{X}}} = \begin{pmatrix} T_{1,1} & T_{1,2} & \cdots & T_{1,n_{\boldsymbol{X}}} \\ T_{2,1} & T_{2,2} & \cdots & T_{2,n_{\boldsymbol{X}}} \\ \vdots & \vdots & \ddots & \vdots \\ T_{n_{\boldsymbol{T}},1} & T_{n_{\boldsymbol{T}},2} & \cdots & T_{n_{\boldsymbol{T}},n_{\boldsymbol{X}}} \end{pmatrix} \tag{3.11}$$

Each row of the $\boldsymbol{Q}_{n_{\boldsymbol{T}},n_{\boldsymbol{X}}}$ contains coefficients in the same order as specified in eq.(3.10). We are determined to forecast the $(1+n_{\boldsymbol{T}})^{th}$ row of the matrix, making use of the previous rows. If such operation is performed accurately, it is also possible to perform the one step ahead forecast of the input series by the IDWT of such estimated row. It turns out that not all the coefficients must be estimated, but some of them can be deterministically calculated, as explained next.

Regarding the coefficients to be estimated, when plotting the columns of $\boldsymbol{Q}_{n_{\boldsymbol{T}},n_{\boldsymbol{X}}}$, some of its column series exhibit regular behavior and thus are easily predictable, for example using neural estimation; others are less regular. Hence using some previous research experience on the analysis and forecast of non-linear non-stationary time series, we deploy one self organizing layer for each of those details columns. A more accurate description of the two methods is given in the next subsections, whereas an algorithm for the virtual prediction of the coefficient is shown in Fig. 3.1.

3.1.1 Fast wavelet transform shift variance

As anticipated above, not every coefficient of the forecasting series has necessarily to be estimated. In fact, shifting and inserting values in $\boldsymbol{X}$ causes the successive forward transforms to possess some shifting features such that the majority of the coefficients of the DWT are automatically determined, hence greatly enhancing the FWT algorithm efficiency applied to streaming datasets and opening a new pathway to the research on the use of discrete wavelet transforms for prediction purposes.

Proposition 3.1. *Given the choice of wavelet filters $\boldsymbol{h}$, $\boldsymbol{g}$, having size $n_{\boldsymbol{h}}$, a source vector $\boldsymbol{X}$ with size $n_{\boldsymbol{X}}$, denoting $M \in \mathbb{N}$ the maximum FWT recursion depth, the size n_{B_m} of each forward*

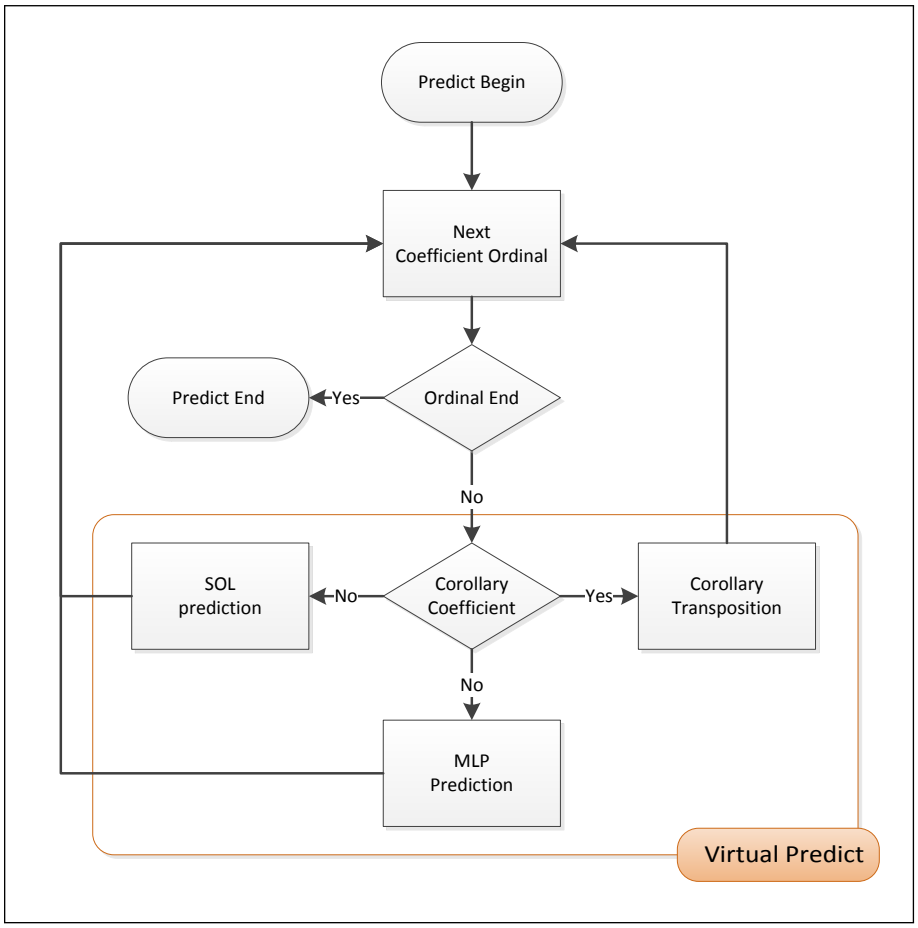

Figure 3.1: Virtual Prediction of DWT coefficients flow chart. At each step, an iterator points to the ordinal of the coefficient to be forecasted. If the theorem defines the pointed coefficient ordinal, the prediction is just a transposition of an older coefficient. Otherwise, a neural or self organizing predictor is choosen. In our studies we chose the neural predictors for series that have a higher degree of autocorrelation, while less autocorrelated series are predicted by the SOLs.

transform subband B_m at recursion depth $m>0, (m\in\mathbb{Z})$ is expressed by the following:

$$n_{B_m} = \frac{n_{\boldsymbol{X}}}{2^m} \tag{3.12}$$

Proof. At each depth m, the size of the input series is halved by 2. Also, the size of the MRA equals the size of the input series. □

Proposition 3.2. *Given a sampled pattern $\boldsymbol{X}$, on which the FWT is performed after having shift-inserted a new value, and given the choice of wavelet filters $\boldsymbol{h}$, $\boldsymbol{g}$, of size $n_{\boldsymbol{h}}$, the number of varying (affected) coefficients at recursion depth $m>0, (m\in\mathbb{Z})$ are the last v_m coefficients of the subband B_m, their number obtained evaluating the following system:*

$$\begin{cases} v_1 = \frac{n_{\boldsymbol{h}}}{2} \\ v_m = (1+v_{m-1})\ div 2 + (1+v_{m-1})\ mod 2 + v_1 - 1 \end{cases} \tag{3.13}$$

Proof. At recursion depth $m=1$, the number of variant coefficients v_1 equals the number of inner products (of the filters $\boldsymbol{h}$, $\boldsymbol{g}$ on the source series), performed on those series subsets changed by the last 2 data insertions. The ordinal of the source series on which the filters are applied when performing the last *full* inner product (i.e. before running into the input series borders) is clearly $n_{\boldsymbol{X}} - n_{\boldsymbol{h}}$. To complete the convolution the filters must advance (with a translation step $n=2$) over the end of the input series, so $\frac{n_{\boldsymbol{h}}-2}{2}$ inner products are left. The total inner products performed over the last coefficients of the series are then $1+\frac{n_{\boldsymbol{h}}-2}{2} = \frac{n_{\boldsymbol{h}}}{2}$. After 2 insertions at level $m=0$ the filter $\boldsymbol{h}$ created a series at level $m=1$ having the last $n_{\boldsymbol{h}}/2$ elements modified (same as explained above for the variant v_1). If the series was not shifted, thus just modifying again the last 2 values of the series at $m=0$, this would imply that the last $n_{\boldsymbol{h}}/2$ coefficients at level $m=1$ would be modified again. But 2 more values at level $m=0$ are inserted instead, and a double shift at level m requires that the series at $m+1$ is shifted once. Hence the filter $\boldsymbol{h}$ creates another c_1 series at level $m=1$ which differs from the previous by $1+n_{\boldsymbol{h}}/2$ elements: 1 element is the first of the previously changed (and shifted) values. This algorithmic behavior is extended to any filter size $n_{\boldsymbol{h}}$. But the number of the varying coefficients v_m at depth m, created by filter $\boldsymbol{g}$, depends on the maximum translations of the filter $\boldsymbol{g}$ on the c_m coefficients that have changed. Generally, denoting v_c the number of such changed coefficients of the c series (which can be an even or odd integer quantity), the maximum translations of a filter of size $n_{\boldsymbol{h}}$ over v_c coefficients, (given the same translation step $n=2$) is trivially $n_{\boldsymbol{h}}/2+\lceil v_c/2\rceil-1$. At depth $m=2$ this means $n_{\boldsymbol{h}}/2+\lceil\frac{1+v_1}{2}\rceil-1 = v1+\lceil\frac{1+v_1}{2}\rceil-1$. But the sum $(1+v_1)\,\mathrm{div}2+(1+v_1)\,\mathrm{mod}2$ is nothing but the definition of ceiling of $\frac{1+v_1}{2}$. Finally, at depth $m=3$, the number of translations of the filters over v_2+1 coefficients is $n_{\boldsymbol{h}}/2+\lceil\frac{1+v_2}{2}\rceil-1 = v1+\lceil\frac{1+v_2}{2}\rceil-1$, and since the equivalence of the ceiling approximation to the sum of the first two terms of the second equation of system (3.13), the proof is obtained by induction. □

Proposition 3.3. *Having a matrix $\boldsymbol{Q}_{n_T,n_X}$ of forward FWTs, each performed at a new shift-insertion in the source series, the non-varying coefficients $d_{h,i}$ at i^{th} column of the B_m sub-band of the h^{th} DWT are equal (and can be retrieved) from the respective rightish $i+1^{th}$ column of a previous DWT $\in \boldsymbol{Q}$ as expressed below:*

$$d_{h,i} = d_{h-2^m,\,i+1}, \quad 0 \leqslant h < n_{\boldsymbol{T}},\ n_{\boldsymbol{T}} > 2^m \tag{3.14}$$

Proof. Since at depth m the series has been subsampled m times, 2^m new insertions are needed to shift by 1 the wavelet coefficients at that depth. □

Theorem 3.4 (On the shift variance of the fast wavelet transform of streaming univariate datasets)**.** *Given an appropriate choice of wavelet filters* $\boldsymbol{h}$, $\boldsymbol{g}$, *having size* $n_{\boldsymbol{h}}$, *a matrix* $\boldsymbol{Q}_{n_T,n_X}$ *of forward FWTs, each performed at a new shift-insertion in the source series, only the last* $v_m = (1+v_{m-1})\ div2 + (1+v_{m-1})\ mod2 + v_1 - 1$ *coefficients of subband* B_m *are affected by one shift-insertion, while the others are previously determined and can be retrieved transposing the respective rightish column coefficients contained in the DWT performed* 2^m *insertions before* ($d_{h,i} = d_{h-2^m,i+1}$).

Proof. Relies on the previous propositions' proofs. □

Remark 3.5. *Tab.3.1 contains the number of variant coefficients* v_m *- also called non-SVT coefficients - for each subband* B_m, *and for different Daubechies filter sizes.*

Theorem 3.6 (On the computational efficiency of the fast wavelet transform applied to univariate streaming datasets)**.** *Given an appropriate choice of wavelet filters* $\boldsymbol{h}$, $\boldsymbol{g}$, *and a matrix* $\boldsymbol{Q}_{n_T,n_X}$, *the following expresses the number of operations* O_c *needed to perform the decimated DWT:*

$$O_c = (2n_{\boldsymbol{h}} - 1)\,(n_{\boldsymbol{X}} + \sum_{m=1}^{M} v_m) \tag{3.15}$$

Proof. The maximum MRA recursion depth M is furnished by eq.(3.7). At each recursion step m and for each filter, $n_{\boldsymbol{B}_m}(p+d)$ operations are performed, where p denotes the number of products, d the number of additions. Both p and d depend on the filter size. Each inner product needs a number of products $p = n_{\boldsymbol{h}}$, and a number of additions $d = p-1$. Hence the total recursion step operations is $2n_{\boldsymbol{B}_m}(n_{\boldsymbol{h}} + n_{\boldsymbol{h}} - 1) = 2n_{\boldsymbol{B}_m}(2n_{\boldsymbol{h}} - 1)$, while the total computations $O = (2n_{\boldsymbol{h}} - 1)\,2\sum_{m=1}^{M} n_{B_m}$. Applying the shift variance theorem (streaming input univariate dataset), this can be rewritten as $O_c = (2n_{\boldsymbol{h}} - 1)\,(\sum_{m=1}^{M} n_{B_m} + \sum_{m=1}^{M} \not{v}_m + v_m) = (2n_{\boldsymbol{h}} - 1)\,(n_{\boldsymbol{X}} + \sum_{m=1}^{M} v_m)$. In the latter eq., non-varying coefficients $\not{v}_m$ are discarded since the theorem 3.4 deployment allows to retrieve them, hence no operations are performed. □

Corollary 3.7 (Asymptotic computational efficiency of the fast 1D wavelet transform of streaming datasets.)**.** *Deploying the shift variance theorem of the fast wavelet transform (thus skipping the unnecessary calculations of SVT coefficients), the computational complexity of the forward FWT of 1D streaming series is reduced by 50% as* $n_{\boldsymbol{X}}$ *tends to infinity:*

$$\lim_{n_{\boldsymbol{X}} \to +\infty} {}^{O_c}/_{O} = {}^{1}/_{2}, \quad n_{\boldsymbol{h}} \ll n_{\boldsymbol{X}} \tag{3.16}$$

Proof.

$$\frac{O_c}{O} = \frac{(2n_{\boldsymbol{h}} - 1)\,(n_{\boldsymbol{X}} + \sum_{m=1}^{M} v_m)}{2(2n_{\boldsymbol{h}} - 1)\,n_{\boldsymbol{X}}} = {}^{1}/_{2} + \frac{\sum_{m=1}^{M} v_m}{2n_{\boldsymbol{X}}},$$

$$\lim_{n_{\boldsymbol{X}} \to +\infty} {}^{O_c}/_{O} = \lim_{n_{\boldsymbol{X}} \to +\infty} {}^{1}/_{2} + \frac{\sum_{m=1}^{M} v_m}{2n_{\boldsymbol{X}}} = {}^{1}/_{2} + \lim_{n_{\boldsymbol{X}} \to +\infty} \frac{\sum_{m=1}^{M} v_m}{2n_{\boldsymbol{X}}} =$$

$${}^{1}/_{2} + \lim_{n_{\boldsymbol{X}} \to +\infty} \frac{\frac{n_{\boldsymbol{h}}}{2} + \sum_{m=2}^{log_2 n_{\boldsymbol{X}} - \lceil log_2 n_{\boldsymbol{h}} \rceil} v_m}{2n_{\boldsymbol{X}}} = {}^{1}/_{2}.$$

□

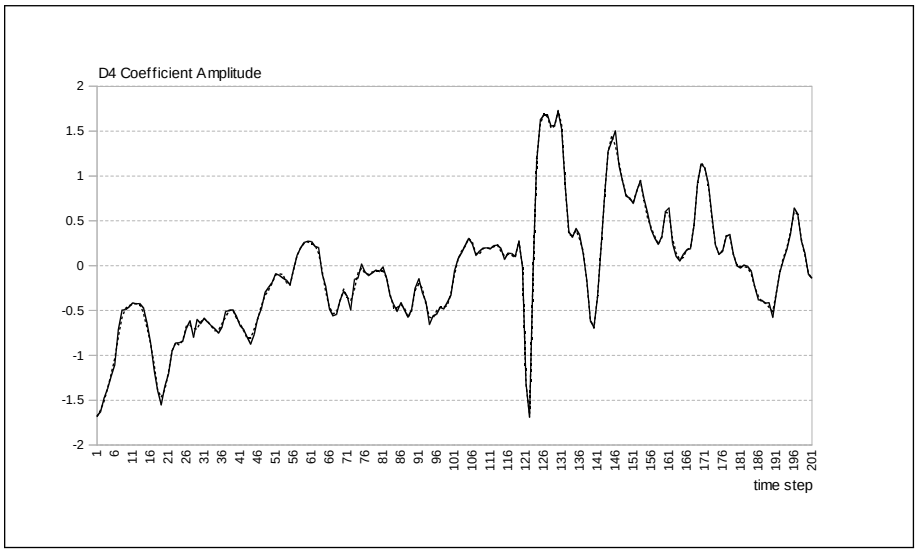

Figure 3.2: Prediction session of one of the higher resolution DWT coefficients by means of a self organizing layer. Dark solid line: forecasted series; dotted line: real coefficients amplitude. The observed forecasting accuracy can be considered quite satisfactory.

3.1.2 Switching virtual predictors

Multilayered Perceptrons

For each of the columns of matrix $\boldsymbol{Q}$ of the subbands coding the signal at coarser resolutions, we train, validate and test a 3 hidden layers backpropagation multilayered perceptron - BPMLP, with single output. The chosen neuronic activation function of the perceptrons is the hyperbolic tangent. If we denote n_L the networks input layer size, the last $1+n_L$ elements of the i^{th} matrix column are preprocessed, in order to create the effective input vector $\boldsymbol{I}_i$ as follows:

$$\boldsymbol{I}_i = \left\{ T_{h,i} - T_{h-1,i} : n_{\boldsymbol{T}} - n_L \leqslant h \leqslant n_{\boldsymbol{T}} \right\} \tag{3.17}$$

hence generating the input vectors to the BPMLPs with the first differences of subsequent coefficients belonging to the same matrix column. Fig.3.2 shows a coefficient prediction session (200 consecutive attempts) performed using the herein described neural regressors.

Self Organizing Layers

In order to perform the forecast of the higher Q column series, we deploy growable self organizing layers, trained sliding the training series with a window of fixed length. The layer size grows every time the focused pattern cannot be recognized unless committing an error greater than a maximum threshold parameter, that we call "granularity" (see Sec. 4.2). If the training series is sufficiently large, the self organizing layer is capable to effectively forecast the one step ahead value, first testing the distance of an input pattern against the stored prototypes, in order to detect the best matching one; then using the information contained in the internal state of the prototype itself. Extensive tests on the DWT coefficients columns showed that this method, when applied to the higher resolutions' coefficients, could outper-

form the forecasting accuracy of the multilayered perceptrons (see Fig. 3.2). The same was not true when applied to the coefficients at coarser resolutions.

3.1.3 Predictors fitness and coefficients optimization

The above postulated shift variance theorem implies that a single variation to the last element of the source series, candidate to be transformed, does not affect the SVT coefficients. Also, let us suppose to compare sets of DWTs of non-shifted series, such that they differ from each other only for the last $l \triangleq x_0$ value: one can discover that each varying coefficient v_{m_i} is governed by its own i^{th} linear relation:

$$v_{m_i} = \alpha_i l + \beta_i \tag{3.18}$$

and consequently it is possible to retrieve, for each non-SVT coefficient ordinal i, the α_i and β_i (slope, intersection) parameters. As previously described, in the herein proposed predictor system a forecast is performed using a hybrid approach, taking advantage both of the novel shifting features for deterministic coefficients retrieval and implementing high precision predictors for non SVT coefficients. However, having found the (3.18), more can be done to improve the forecast of the one step ahead DWT. In fact, having queued the previous forward transforms and their forecasts in matrix Q, a fitness value for each predictor of the varying coefficients (hence excluding SVT coefficients) can be retrieved, measuring their absolute forecasting error. Such evaluation allows the system to select the predictor that achieved the best accuracy, and rely on it for the next forecast, calculating its hypothetical l. Indicating l_f the l calculated for the best performer predictor at ordinal f, such value allows us to determine a maximum probability on-fitness estimate of the others non-SVT coefficients:

$$\begin{cases} \hat{l} \triangleq l_f \\ v_{m_i} = \alpha_i \hat{l} + \beta_i \end{cases} \tag{3.19}$$

Deploying equations' system (3.19) further improves the forecasting accuracy of an MRA-assisted machine learning predictor.

3.1.4 Inverse Discrete Wavelet Transform for prediction

The last operation needed to extract the forecasted value is to perform an inverted transform of the forecasted forward DWT. The forecasted one-step ahead value is obviously the last element of the inverted forecast:

$$\widehat{T} = \left\{ w_i : 0 \leqslant i \leqslant n_{\boldsymbol{X}} - 1 \right\} \tag{3.20}$$

$$\widehat{\boldsymbol{X}} = \widetilde{T}\,\widehat{T} \tag{3.21}$$

$$\hat{x} = \widehat{\boldsymbol{X}}_{n_{\boldsymbol{X}}-1} \tag{3.22}$$

where $\widetilde{T}$ denotes the inverted DWT.

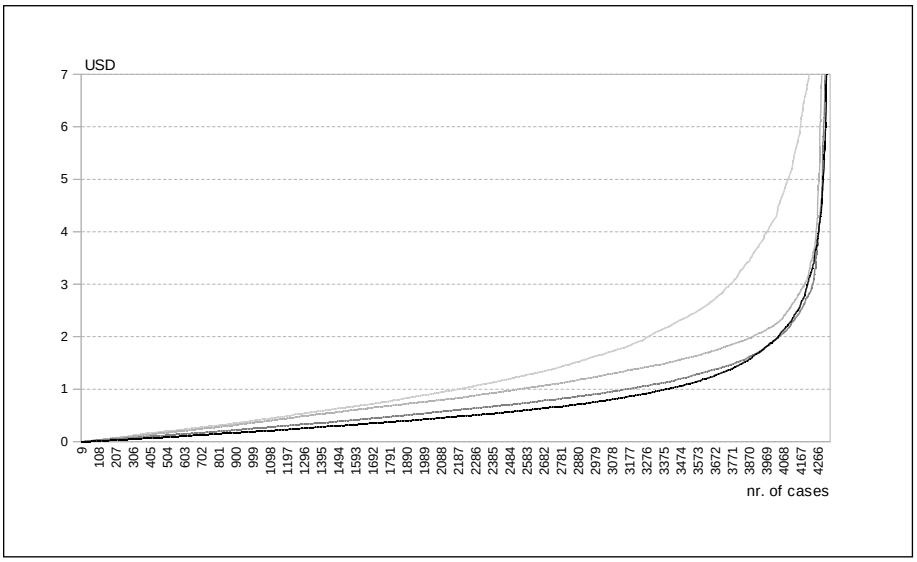

Figure 3.3: Sorted absolute errors of the Bitcoin hourly price forecast session. Light line: naive prediction, the worst performer among the benchmark methods; dark lines: performance of clustering self organizing layers prediction system; the better performer is featured by multiscale pattern recognition and cooperating neural predictors; black line: performance of the fast wavelet transform assisted predictor, herein proposed. The results show the novel system outperforms all the benchmark models.

3.1.5 System's update and retrain operations

The incoming data is shift-inserted in the source series, and a new DWT is immediately performed and stored in a history queue. This is useful to determine the predictors fitness value at the previous forecasting step. Also, the confirmed value is used to retrain both the neural and self organizing layer type predictors.

3.2 Results and discussion

Since the beginning of the work we were interested in developing a system whose results could be directly compared using one of our previous works as a benchmark model. To this purposes, we selected the Bitcoin-USD hourly series (close price) spanning from January 1st 2013 to June 30th 2015, whose description can be found in Sec. 2.2. The whole dataset is partitioned into an in-sample period, used for training and validation of the system, (from January 1st 2013 to December 31st 2014), and an out-of-sample period (from January 1st 2015 to June 30th 2015) to test the system on real data. Price forecast session results are plotted in Fig.3.3 (sorted absolute errors for the Bitcoin hourly price prediction), showing that the system substantially outperforms the benchmark methods, including the predictor system proposed in the previous chapter. Occurrence of errors less than 50 cents is 51%, while the best performer among the benchmark methods achieves 43%. The system cumulative absolute error is 3259 USD, while the best performer benchmark's one is 3662 USD.

We also tested the engine other time series, made publicly available by the Eurostat. They are monthly indicator series of the Eurozone Producers Price Index and, as such, they generally possess a higher autocorrelation than financial price series. Industrial output PPI series

	MAE			RMSE			MAPE		
	MLP	SVM	SYS	MLP	SVM	SYS	MLP	SVM	SYS
Industry[a]	3.4	3.2	**2.9**	5.7	4.4	**4.1**	3.6	3.4	**3.1**
Int.goods[b]	2.5	3.1	**2.5**	3.8	4.3	**3.4**	2.7	3.4	**2.5**
Mining[c]	4.8	4.4	**3.9**	7.3	5.7	**5.2**	4.8	4.4	**3.9**

Table 3.2: Prediction performance of Producers Price Indexes - PPI time series Euro Area 19, Monthly data. Accuracy is expressed in terms of mean absolute error (MAE), root mean square error (RMSE), mean absolute percentage error (MAPE). Values x 10^{-1}. a) Industry, 1981-2015. b) Intermediate Goods, 1991-2015. c) Mining, quarrying; manufacturing; electricity, gas, steam and air conditioning supply, 1991-2015. Source: Eurostat.

spans from 1981 to 2015 (420 samples), which are divided into an in-sample period used for training (the first 180 observations) and an out-of-sample period (the remaining 240 observations). Intermediate goods and mining output PPI series span from 1991 to 2015 (300 samples): the in-sample and out-of-sample periods are respectively 180 and 120 samples. The results are shown in Tab.3.2): Industry, Intermediate goods, and Mining-quarrying series prediction performance is benchmarked against a triple hidden layer backpropagation MLP predictor (self implemented) and support vector machines - SVM (Dlib-ml implementation by [29]). The performance of the system against the benchmark methods is evaluated in terms of mean absolute error, root mean square error and mean absolute percentage error (MAE, RMSE, MAPE). Generally the system outperforms both the MLPs and SVMs, suggesting that the forecasting accuracy is quite satisfactory.

3.3 Further developments

The use of the novel shift variance theorem of the DWT allows to greatly reduce the extensive testing times of the proposed system, and it provides the predictor a very high absorption rate. In fact, since very few coefficients must be estimated, only a small number of predictors is required. For example, if the input series size $n_X = 256$, using a $N = 4$ (8 taps) Daubechies filter, hence $M = log_2 256 - \lceil log_2 8 \rceil = 5$, (see eq.3.7) the total number of non SVT coefficients per single DWT (hence those to be estimated) is just 31. Also, the correct retrieval of deterministic DWT coefficients allows to obtain a much more accurate estimated one-step ahead crystal.

Regarding the system training operations, it is worth noting that long datasets availability allows to train the self organizing layers in a very effective way. SOLs can learn either submitting patterns sampled at constant interval scales, (hence capturing time series multiscale properties that are not immediately evident to the human eye); alternatively, submitting consecutive patterns of fixed size. In the presently illustrated system the single pattern approach is used to train SOL predictors to the wavelet coefficient queues of the highest subbands of the Bitcoin price window's DWT. During the analysis of the Eurostat series, the efficacy of SOLs is tested scarce, the reason being the shortness of the input datasets and hence of the unavailability of long DWT coefficient queues; consequently the predictors are built using only artificial neural networks.

Relatively to the training of the neural predictors (multilayered perceptrons in the present implementation of the system), we emphasize the importance of the input data preprocess-

ing operations on the forecasting accuracy achievable. In the case of price series prediction, the creation of first difference input patterns as illustrated by the (3.17), greatly enhances the forecasting results and reduces the number of retrains of the MLPs during the update operations (meaning that the achieved precision is satisfactory). During the Eurostat series analysis instead, we utilize unpreprocessed data input patterns since the aforesaid Statistical Institute already provides data series in percentage and no further manipulation is considered necessary.

Should the system be implemented to perform the prediction of non-normalized time series, the type of input data preprocessing to be performed should be carefully considered.

The present system was initially developed with the aim to serve as a general time series predictor and thus no *a priori* assumptions on the input series features were made during the code implementation. When used to forecast the Bitcoin - US Dollar exchange rates (hourly data), the results are encouraging: a raw estimate of the DWT coefficients, fine tuned using the linear properties outlined in the Subsec.3.1.3, greatly improves the prediction accuracy of the series. Similarly, the prediction tests on statistical time series other than the financials outperformed the benchmark methods, suggesting that the system has a good degree of generalization.

Given the interesting achievements related to the shift variance properties of the fast wavelet transform for 1D streaming datasets, it is important to test the system on other types of time series, for example those generated by IoT sensors and devices. A confirmation of the forecasting accuracy of the proposed system could lead valuable commercial or industrial applications.

Finally, further research could be devoted to explore the shift variance properties of the decimated discrete wavelet transform of 2D streaming time series, such as those generated by probes, drones and, generally, moving observers.

Chapter 4

Intrinsic mode decomposition assisted machine learning frameworks

In the recent years several reassignment techniques are capturing the attention of researchers in the field of signal analysis. Among the possible reallocation methods the Synchrosqueezing approach allows to map a Continuous Wavelet Transform - CWT from the time scale to the time frequency plane. Its mathematical foundations prove that the Synchrosqueezing transform - SST can be directly related to the Empirical Modes Decomposition, since both methods allow the decomposition of a signal having finite energy in its intrinsic mode functions, each having time-varying frequency and amplitude. We develop a fast implementation of the Synchroqueezed Wavelet Transform suitable for the synchronic extrusion of instantaneous frequency information from univariate time series. We plan the development of non parametric estimators or neural regressors to analyze and predict the time series using the SST as a preprocessing module to assist the machines in learning the time series features.

Recently the state of the art relative to the prediction of time series is pointing towards novel approaches based on signal decomposition or time-frequency analysis. The first approach, developed in the late 90' by Huang N. [24] and named Empirical Mode Decomposition, allows the decomposition of a signal in a fixed number of intrinsic mode functions - IMFs, which maybe viewed as component series of the form $x(t) = A(t)cos(\phi(t))$, plus a residual series. Such decomposition can be considered as a generalized Fourier series, in which the components phases and amplitudes are not constrained to be constant. Interest in such approach motivated Wu and Huang [50] to further research innovations, that brought them to formulate the Ensemble EMD - EEMD, in which the IMFs of a series are found after having perturbated the signal with gaussian noise; the result is an improved noise-assisted data analysis for IMF identification, at a slightly higher computational cost.

The wide interest in EMD and EEMD and their absolute generalized utility motivated researchers to explore the possible mathematical foundations involved in the aforesaid empirical approaches. Recently, Daubechies, Lu and Wu [12], starting from the definition of the CWT and after having provided the algorithm to perform its reassignment, the Synchrosqueezed Continuous Wavelet Transform - SST, proved that the latter can be viewed as an adaptive time-frequency decomposition whose intent is the same as the EMD. The

mathematical rigorousness of the novel approach brought, more recently, Thakur, Brevdo et al. [47] to provide insights on the stability of Synchrosqueezing, and implementation aspects related to the decomposition and reconstruction of sampled series via the SST. Our interest in the SST approach is motivated by the possibility to contrive a prediction system making use of the Synchrosqueezing transform as a dataset preprocessing module, as well as by new possible developments in the machine learning field. Yet, not many works focused on the use of the CWTSST to prediction purposes can be found, the reason being the novelty of such model. This is the main argument of the present chapter and it will be described in the next sections. We are planning to project and develop a prediction system suitable for the analysis of streaming datasets such as the one tested in the previous works, or other time series such as the search engine query trends, whose prediction importance goes beyond the academic environment and, as such, could lead to valuable commercial or industrial applications.

This chapter is organized as follows: Sec. 4.1 provides the theoretical foundations of the SST, the principal technical aspects related to the SST implementation, as well as the preliminary results obtained. Sec. 4.2 provides a description of the possible machine learning approaches suitable to perform a time series forecast using the time series decompositions of the CWTSST. The Sec. 4.3 provides conclusions and portrays further research directions. The interested reader can also refer to Sec. 4.4.1 for the mathematical background relative to the CWT in frequency domain, Sec. 4.4.2 for the time derivative of the CWT, and Sec. 4.4.3 for theoretical background of the signal reconstruction after SST decomposition.

4.1 Synchrosqueezed Wavelet Transforms

A Continuous Wavelet Transform - CWT allows the time-varying analysis of the frequency content of a signal $f(t)$ having finite energy. Choosing an appropriate mother wavelet function $\psi(t)$ such that its Fourier transform $\hat{\psi}(\xi) = 0$, $\xi < 0$, the CWT of $f(t)$ is defined as a series of parametric convolutions of the signal and the complex conjugate of scaled versions of the wavelet. Note that a wavelet function conforming the above mentioned features is complex [14,35]. Having performed the CWT, reconstructing the signal means integrating, in the scale direction, the convolutions of the transform coefficients and the scaled versions of the wavelet function:

$$W_f(a,b) = \int_{-\infty}^{\infty} f(t)\, a^{-1/2}\, \overline{\psi\left(\frac{t-b}{a}\right)}\, \mathrm{d}t. \tag{4.1}$$

$$f(t) = C_\psi \int_{-\infty}^{\infty} \int_{0}^{\infty} W_f(a,b)\, a^{-5/2}\, \psi\left(\frac{t-b}{a}\right) \mathrm{d}a\, \mathrm{d}b, \tag{4.2}$$

where C_ψ is constant and depends only on the wavelet. Eq. 4.1 differs from the Discrete Wavelet Transform - DWT, in which the parameters a, b are selected from a discrete sublattice [10]. Performing the CWT of a pure tone it can be observed that the frequency localization is spread out [12] along the scale axis, and that such effect cannot be avoided. Specifically, considering a tone $f(t) = A\cos(\omega t)$ and supposing that the chosen wavelet has a spectrum $\hat{\psi}(\xi)$ concentrated in proximity of $\xi = \omega_0$, the above mentioned spreading effect

would be around the scale point $a = \omega_0/\omega$. In order to provide a much more definite instantaneous frequency detection of a signal, however, it can be noted that if the chosen wavelet is complex, the real and imaginary components of the CWT contain enough phase information to pinpoint the oscillatory behavior of $f(t)$ in b. Hence the intuition [14] to retrieve a matrix $\omega_f(a,b)$, associated to the $W_f(a,b)$, containing the instantaneous frequencies extracted from the CWT via phase transformation:

$$\omega_f(a,b) = -i\frac{\partial/\partial b\, W_f(a,b)}{W_f(a,b)} \tag{4.3}$$

Where $\partial/\partial b\, W_f(a,b)$ can be calculated using the time derivative property of the Fourier transform (the interested reader can refer to Sec. 4.4.1 and Sec. 4.4.2).

Possessing both W_f and ω_f matrices it is now possible to reassign the wavelet transform, in order to pass from a time-scale representation to a time-frequency plane. Let us denote $S(W,\omega)$ the Synchrosqueezing operator, and $T_f(\omega,b)$ the Synchrosqueezed Wavelet Transform of $f(t)$, such that:

$$\begin{gathered} S(W_{a,b},\omega_{a,b}) : (a,b) \to (\omega_{a,b},b) \\ T_f(\omega,b) = \int W_f(a,b)\, a^{-3/2}\, \delta(\omega(a,b)-\omega)\, \mathrm{d}a, \quad a : W_s(a,b) \neq 0 \end{gathered} \tag{4.4}$$

In a discrete environment, ω spaces linearly or logarithmically from the fundamental to the Nyquist frequency of a sampled series. If one chooses a linear frequency scale, as we do in the present work, having a vector of frequencies $\boldsymbol{\omega} = \{\omega_i : 1 \leqslant i \leqslant N\}$, the operator $S(W,\omega)$ can be contrived simply using a standardized lower bound algorithm to search the ω_i, (center frequency of a "bin" gathering a group of frequencies $[\omega_i - \Omega, \omega_i + \Omega]$, $\Omega = (\omega_i - \omega_{i-1})/2$), nearest to an instantaneous frequency $\omega_f(a,b)$. Once the destination bin is found, the contribution of the $W_f(a,b)$ can be summed into the right destination position in matrix $\boldsymbol{T}_f$:

$$T_f(\omega,b) = 1/2\Omega \sum_k W_f(a,b)\, a_k^{-3/2}\, (\Delta a)_k, \quad a_k : |\omega(a_k,b) - \omega| \leqslant \Omega \tag{4.5}$$

The Synchrosqueezing Wavelet Transform possesses a corresponding reconstruction algorithm. Let us denote $f(b)$ the reconstructed signal. In [12] it is proved that:

$$f(b) \propto \int_0^\infty W_f(a,b) a^{-3/2} \mathrm{d}a \tag{4.6}$$

The interested reader can also see a proof rewritten in a more pedagogical fashion in Sec. 4.4.3. The discrete version of the reconstruction algorithm can then be written from eq.(4.5):

$$f(b) \approx \Re\Big\{C_\psi^{-1} \sum_i T_f(\omega_i,b)(2\Omega)\Big\} \tag{4.7}$$

The above descriptions highlight the foremost theoretical aspects related to the SST, retained as foundation to the development of an advanced Synchrosqueezed Transform implementation.

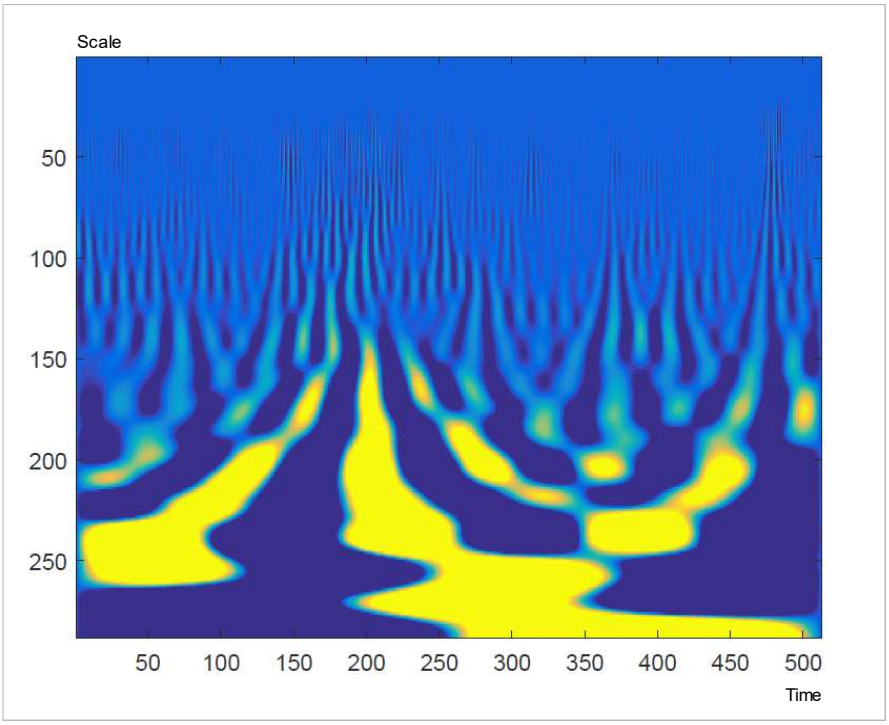

Figure 4.1: Continuous Wavelet Transform of a Bitcoin price session, imaginary component, Morlet wavelet ($\sigma = 2\pi$), 512 hourly rates beginning from 1 January 2016 (scales in the vertical axis, time-steps in the horizontal axis). The smearing effect along the scale axis reproduces the *spread out* issue described by Daubechies in its dissertations, particularly referencing to exemplar pure tones, (see [12]).

4.2 Method

After having reassigned the CWT according to the procedure described in Sec. 4.1, the vector of frequencies $\boldsymbol{\omega}$ can be partitioned in equivalence classes having a chosen bandwidth, and the reconstruction described by eq.(4.5) is performed piecewise, thus creating the set of intrinsic mode functions $m_i(t)$ meant to be the original series components. From each $m_i(t)$, instantaneous amplitudes and phases can be recovered. Note that this means the SST can be used for prediction purposes. Also, it is our opinion that such transform paves the way to novel machine learning approaches that focus on learning the rates of change of time series instead of fitting their values; such approaches could easily become a very active area of research in the next few years. Our present position is to explore the possible machine learning approaches founded on preprocessing the input series via the SST, and forecasting the one step ahead real value of the single component series using specialized estimators. In the next section we explore the risks connected to different prediction realizations, using both traditional methods and the novel techniques that we are researching.

For the scope of the present demonstration we develop a fast C++ implementation of the CWTSST algorithm, in which the complex wavelet function $\psi(t)$ can be selected via parametric polymorphism. In order to be able to effectively use the system for prediction purposes, given the heaviness of the stepwise calculations, efficiency is considered paramount.

Several time series are downloaded, while the Bitcoin hourly exchange rates (BTCUSD), available from the previous research, are used as well for automated testing and debugging

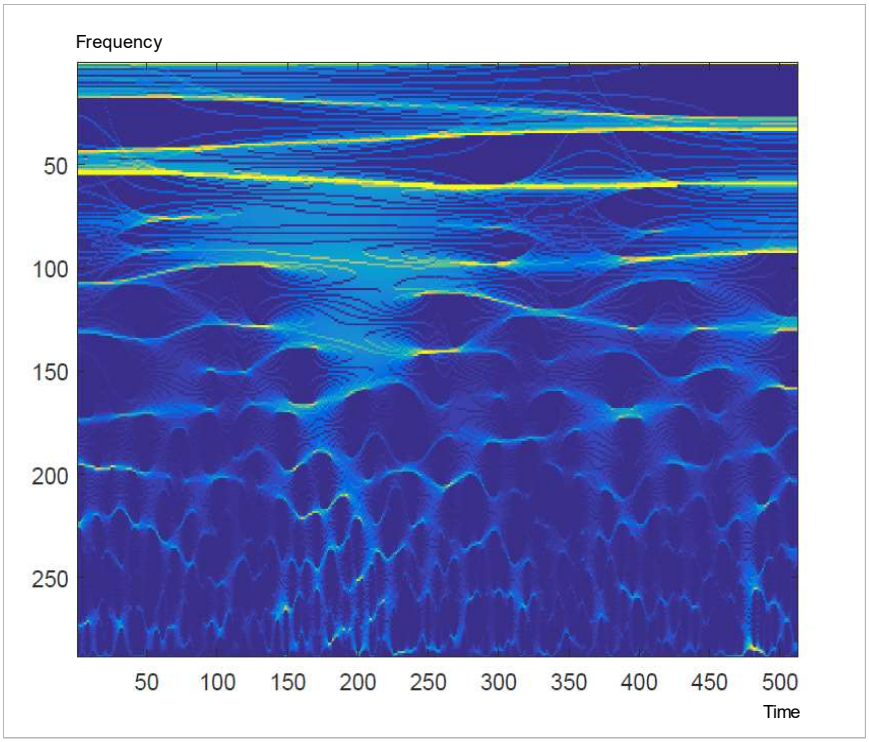

Figure 4.2: Synchrosqueezed Wavelet Transform of the same Bitcoin price session as in Fig. 4.1, magnitudes of the complex values. Note that the two representations are flipped vertically against each other, this is correct since CWT scales and SST instantaneous frequencies are inversely proportional. Despite such optical inconvenient though, it is not difficult to appreciate the precision of the SST frequency extrusions, as depicted in yellow in the upper part of the plot. Both the lower and higher frequencies of a difficult-to-predict series such as the Bitcoin exchange rates are thus easily detected using the SST, promising new approaches to effective time series forecasts.

purposes.

We perform extensive analysis and synthesis tests on the gathered datasets in order compare the reconstruction accuracies. Finally we are going to choose the wavelet function which gives the lower reconstruction error results. The system must be able to read streaming datasets such as the above mentioned search trends chunked with a window of fixed length, the latter sliding from the beginning of the series to the last known data sample. The CWTSST can be performed in a three step fashion:

1. Continuous Wavelet Transform - CWT of the candidate data sequence;
2. Phase Transform - PT of the CWT;
3. Reassignment - Transfer of the CWT from the time-scale to the time-frequency plane.

4.2.1 Continuous Wavelet Transform

The selected input pattern $\boldsymbol{X} = \{x_i : 0 \leqslant i \leq N\}$, is first preprocessed before being analyzed via the CWT. If $N < 2^j$, $j \in \mathbb{Z}$, an integer $N_p = 2^{(1+log_2 N)}$ is chosen as the size of a pattern $\boldsymbol{X}'$. Since $N_p > N$, the extra vector space is used to symmetrically pad the payload content. Such padding procedure is useful to reduce the negative effects of well known border problems when performing the CWT. The Synchrosqueezed Transform is calculated implementing the algorithms described by eq. (4.4).

In Fig. 4.1 we depict the first tests of the newly implemented CWT, showing the imaginary component of the CWT of a Bitcoin price session, calculated instantiating the model classes with the Morlet wavelet. Let us recall the Fourier transform of such wavelet class:

$$\begin{cases} \hat{\psi}(\xi) = c_\sigma \pi^{-1/4} \left(e^{-1/2 (\sigma - \xi)^2} - k_\sigma e^{-1/2 \xi^2} \right) \\ c_\sigma = \left(1 + e^{-\sigma^2} - 2 e^{-3/4 \sigma^2} \right)^{-\frac{1}{2}} \\ k_\sigma = e^{-1/2 \sigma^2} \end{cases} \tag{4.8}$$

where c_σ and k_σ are clearly constants and depend both on parameter σ. The wavelet function is above written in the frequency domain since both forward and inverse CWT can be expressed in the frequency domain via convolution theorem (see also Sec. 4.4.1). Note that this generally greatly improves the CWT-ICWT algorithms' efficiency since each convolution of eq.(4.1) is reduced to a multiplication (see eq. (4.14)), where:

$$\hat{\overline{\psi}}_a(\xi) = a^{1/2} \overline{\hat{\psi}(a\xi)} \tag{4.9}$$

Eq.(4.9) allows to find, for each scale iteration of the CWT, the Fourier transform of the scaled version of the wavelet. Hence, for example, at each scale a_i the Morlet wavelet of eq.(4.8) can be written as follows:

$$\hat{\psi}(a_i \xi) = a_i^{1/2} c_\sigma \pi^{-1/4} \left(e^{-1/2 (\sigma - a_i \xi)^2} - k_\sigma e^{-1/2 a_i^2 \xi^2} \right) \tag{4.10}$$

Note that, in case of streaming datasets (such as the case of the present research), on which one must calculate the CWT at each data shift-insertion, the set of $\hat{\psi}(a_i \xi)$, $a_0 \leqslant a_i \leqslant a_{max}$ can be calculated once and mapped into a high performance container specialized for fast key-value retrieval, hence greatly enhancing the CWT algorithm efficiency.

4.2.2 Phase Transform

The phase transform of the CWT matrix $W_f(a, b)$ is defined in eq.(4.3) as a matrix of the same size as the CWT's one, containing the partial derivatives of the CWT with respect to space. Such partial derivatives are divided by the wavelet coefficients in order to purge the Phase Transform from the artifacts introduced using the instantiating wavelet. As mentioned above, the wavelet choice should be limited to complex (analytic) wavelets (even if the candidate series to be analyzed is real), since the imaginary part of the complex CWT values contains phase information. The first derivatives of the CWT coefficients could then be found

as follows:

$$\partial/\partial b\, W_f(a,b) = \angle W_f(a,b) \triangleq \arctan \frac{\Re(W_f(a,b))}{\Im(W_f(a,b))} \tag{4.11}$$

The interested reader can also find an alternative procedure in Sec. 4.4.2 eq.(4.16). We implemented both approaches and we found negligible differences in efficiency.

4.2.3 Reassignment procedure

Having calculated both the CWT and the PT matrices, the reassignment of the wavelet transform to the frequency space plane is now possible. Both matrices have the same size, hence they can be iterated in space simultaneously. At each iteration step, the $\omega_f(a,b)$ points the correct frequency bin which the corresponding $W_f(a,b)$ coefficient can be summed in. Such reassignment is performed for each row of the CWT and PT matrices.

The resulting representation in the frequency-space plane is much more concentrated than the CWT, and the smearing effect of the scale-space representation is greatly reduced. This is the feature of the CWTSST that makes possible to perform a prediction using machine learning objects. The synchrosqueezed representation of the same BTCUSD dataset used in Fig. 4.1 is plotted in Fig. 4.2.

4.2.4 Aspects of machine learning used for IMFs prediction purposes

Once the reassignment of the CWT into the frequency-space plane is completed, we are going to partition the frequency axis in a fixed number of sets having the same size, depending on the number of modes $m_i(t)$ we intend to use to decompose the signal. Having created a number N_m of intrinsic modes $m_i(t)$, our goal is to perform a prediction of each $m_i(t)$ one-step ahead. If the IMFs could be used as a representative training set suitable to input our inference modules, then reconstructing the one-step ahead forecast would simply be obtained by summing the single mode estimations[1]:

$$\widehat{f}(t) = \sum_k \widehat{m}_i(t), \quad 0 \leqslant k \leq N_m \tag{4.12}$$

The prediction could be performed using neural regression, for example by means of a back-propagation multilayered perceptron - BPMLP network plugged to each $m_i(t)$. However, designing the prediction system in such way, there would be risks associated to the accuracy that we set to be achieved, since the $m_i(t)$ would exhibit variable oscillatory behavior. Hence in order to train adequately each BPMLP, the input size should be greater or at least equal to one full oscillation of the mode. Note that this means that, first: the BPMLPs could not have the same input layer size (and consequently the configuration of the hidden layers would not be uniform among the neural networks); secondly, in some cases the input size of the network would be huge, causing lengthy training (and retraining) operations and degrading the whole system's efficiency.

[1] let us denote in this case the estimations using the *hat* accent and hope that no confusion with the Fourier transform notation occurs.

Thus in order to anticipate such issues the number of hidden BPMLP layers should be kept low. This should not represent a real limitation since, generally speaking, (and given our previous experience on the development and testing of artificial neural networks) the number of hidden perceptron layers must be necessarily increased only if the MLP is used for classification purposes (i.e. the network must learn to classify a great quantity of differently labeled patterns). In this case the MLP is to be used for regression purposes, hence once the features of a mode $m_i(t)$ have changed, the machine could immediately forget the previous configuration and adapt to the changing statistical properties of $m_i(t)$.

A different approach would be to train a set of N_f detached perceptrons to forecast the one-step ahead instantaneous frequencies of the series. Let us recall that a Synchrosqueezed Wavelet Transform outputs a matrix $T_f(\omega, b)$ of size $\{N_f, N\}$, where N_f is the size of the frequencies vector $\boldsymbol{F}$, and N is the size of an unpadded input series. Partitioning the frequency axis into equally sized segments would eventually decompose $T_f(\omega, b)$ in intrinsic mode functions, whose union is eventually the CWTSST itself. Note that this also guarantees that each single component has a maximum frequency excursion, hence training neural regressors to learn how those frequencies vary is feasible and it is an efficient operation. Each perceptron would be responsible to learn the $\omega_f(a, b)$ rate of change in time direction, i.e. $\frac{\partial}{\partial t}\omega_f(a, b)$, in order to be able to analytically find the one-step ahead $\omega_f(a, b{+}1)$ value having empirically read the previous $\omega_f(a, b)$. Success in implementing the proposed approach is key to machine-learning innovations suitable to predict series analyzed via the SST.

4.3 Further developments

After having investigated the literature related to signal processing, particularly the frequency space analysis approach, a novel reassignment procedure is attracting our attention, as well as the possibility to use it for predicting streaming datasets series. We start from the basic idea that a time series can be sampled in patterns of fixed size; such patterns can be preprocessed in order to produce inputs to a predictor system.

The approach we are going to propose is based on the original series decomposition (time-frequency analysis), based on synchronic instantaneous frequency extrusion operations performed via the SST. Such preprocessing step is meant to assist a group of cooperating statistical predictors or neural networks; our implementation experience will let us face neural regression issues more rapidly, and for such reason we will probably privilege multilayer perceptrons tests first, in order to evaluate the difficulties related to the forecast of oscillatory signals in a reasonable amount of time. Eventually, we may formulate a novel machine learning technique for real-valued forecast, tailored to predict amplitude and frequency modulated signals (such as the intrinsic modes extracted via the SST) in an efficient manner.

We believe that the success of the proposed approach is key to machine-learning innovations suitable to predict several categories of time series (such as the financials series, the search engine query trends series, the streaming datasets produced by IoT sensors and devices, etc.); also, the extensibility of the model to the analysis and prediction of general time series cannot be overruled at this time.

4.4 Appendices

4.4.1 A1: CWT in frequency domain

Let us denote $\widehat{W}_f(a,\xi)$ the Fourier transform of $W_f(a,b)$, as well as $*$ the convolution. Note that each crystal of the $W_f(a,b)$, obtained fixing a, can be seen as a convolution of f and the complex conjugate of a scaled version of wavelet $\overline{\psi_a(t)} = \overline{\psi(-t/a)}$. Let us indicate $z = {}^{-1}/_{a}$, and, recalling the time and frequency scaling property of the Fourier transform, let us rewrite the CWT using the convolution theorem as follows:

$$\begin{aligned} W_f(a,b) &= a^{-1/2}\left[f(t) * \overline{\psi_a(t)}\right] \\ \widehat{W}_f(a,\xi) &= a^{{}^{-1}/_{2}}\,\widehat{f}(\xi)\,\frac{\overline{\widehat{\psi}({}^{-\xi}/_{z})}}{|z|} \end{aligned} \tag{4.13}$$

Finally, substituting z with $a = {}^{-1}/_{z}$,

$$\widehat{W}_f(a,\xi) = a^{{}^{-1}/_{2}}\,\widehat{f}(\xi)\,a\,\overline{\widehat{\psi}(a\xi)} = a^{{}^{1}/_{2}}\,\widehat{f}(\xi)\,\overline{\widehat{\psi}(a\xi)} \tag{4.14}$$

It is important to note that, using eq.(4.14), for each scale iteration of the CWT, a convolution becomes a multiplication. This allows greatly reduce the computational resources needed to perform the CWT (using an implementation suitable to analyze digital signals), thus performing calculations in a much more efficient way.

4.4.2 A2: time derivative of the CWT

Let us recall a time-derivative property of the Fourier transform such that:

$$\widehat{{}^{d}/_{dx}\,f(x)}(\xi) = 2\pi i\xi\,\widehat{f}(\xi) \tag{4.15}$$

Our objective is to find a convenient way to calculate the CWT's time derivative. The latter can be expressed similarly using eq.(4.15):

$$\widehat{{}^{\partial}/_{\partial b}\,W_f}(a,\xi) = 2\pi i\xi\,\widehat{W}_f(a,\xi) = 2\pi i\xi\,a^{{}^{1}/_{2}}\,\widehat{f}(\xi)\,\overline{\widehat{\psi}(a\xi)} \tag{4.16}$$

Finally, ${}^{\partial}/_{\partial b}\,W_f(a,b)$ can be retrieved performing the inverted Fourier transform of the result of eq.(4.16).

4.4.3 A3: Reconstruction after SST

Having expressed the Fourier transform $\widehat{W}_f(a,\xi) = a^{1/2}\,\widehat{f}(\xi)\,\overline{\widehat{\psi}(a\xi)}$, we can rewrite eq.(4.4) substituting $W_f(a,b)$ with the inverted Fourier transform of eq.(4.14):

$$\begin{aligned}
\int_0^\infty W_f(a,b)\,a^{-3/2}\,\mathrm{d}a &= \int_0^\infty \left(\frac{1}{2\pi}\int_{-\infty}^{\infty}\widehat{W}_f(a,\xi)\,e^{i\xi b}\mathrm{d}\xi\right)a^{-3/2}\,\mathrm{d}a \\
&= \int_0^\infty \left(\frac{1}{2\pi}\int_{-\infty}^{\infty} a^{1/2}\,\widehat{f}(\xi)\,\overline{\widehat{\psi}(a\xi)}\,e^{i\xi b}\mathrm{d}\xi\right)a^{-3/2}\,\mathrm{d}a \\
= \left(\int_0^\infty a^{-1}\,\overline{\widehat{\psi}(a\xi)}\,\mathrm{d}a\right)&\left(\frac{1}{2\pi}\int_{-\infty}^{\infty}\widehat{f}(\xi)\,e^{i\xi b}\mathrm{d}\xi\right);\quad \text{setting } a\xi=\omega, \\
\int_0^\infty W_f(a,b)\,a^{-3/2}\,\mathrm{d}a &= \left(\int_0^\infty \overline{\widehat{\psi}(\omega)}\,\frac{\mathrm{d}\omega}{\omega}\right)\left(\frac{1}{2\pi}\int_{-\infty}^{\infty}\widehat{f}(\xi)\,e^{i\xi b}\mathrm{d}\xi\right) \qquad (4.17)\\
&= f(b)\int_0^\infty \overline{\widehat{\psi}(\omega)}\,\frac{\mathrm{d}\omega}{\omega};\quad \text{hence,} \\
f(b) = \frac{\int_0^\infty W_f(a,b)\,a^{-3/2}\,\mathrm{d}a}{\int_0^\infty \overline{\widehat{\psi}(\omega)}\,\frac{\mathrm{d}\omega}{\omega}} &= C_\psi^{-1}\int_0^\infty W_f(a,b)\,a^{-3/2}\,\mathrm{d}a.
\end{aligned}$$

In eq.(4.17), the denominator $\int_0^\infty \overline{\widehat{\psi}(\omega)}\,\frac{\mathrm{d}\omega}{\omega}$ is nothing but the constant generally denoted C_ψ, and it depends only on the wavelet function.

Chapter 5

Implementation aspects

All the system herein proposed is developed as ISO/IEC 14882:2014 conformant C++ implementation compiled with speed maximization optimization, inline functions expansion and intrinsic functions enabled, target machine x64.

The code is featured by heavy use of parametric polymorphism, useful to create general purpose core components and suitable to maximize modules reusability. It takes advantage of some of the features introduced by the aforementioned standard, such as variadic templates, that allow to contrive class generators in which the number of instantiating classes is not known.

5.1 Variadic Template Classes

This particular features enables implementors to design highly flexible template classes, to be instantiated at compile time, thus reducing the number of runtime parameters and maximizing efficiency. The next example should clarify the advantage of parametric polymorphism applied to the generation of a required neural network:

```
namespace artificial_neural_networks
{
        template <class _Functype, // logistic, hyperbolic_tangent, ...
                class _InputLayerType,
                        template <class> class... _LayerTypes>
        class base_multilayer_perceptron; /*undefined*/

        // ...
```

In the code listing above a template basic class suitable to build multilayered perceptrons is declared. Such template class is not defined yet, since the template class body may vary substantially depending on the type of input layer class chosen to instantiate the template. Instead the definition of the class is made through template *specializations*, in which the behavior of the mother template is defined accordingly to the content of some specific features. In this example, by the time we attempted to implement one of the base classes suitable to

build artificial intelligences, we were not aware of the possibility to contrive input layers different from what the state of the art described. For this reason the possibility to instantiate the mother template with more advanced preprocessing-endowed input classes could not be overruled. In the code that follows, we provide an excerpt of the template specialization for general multilayer perceptrons instantiated with a basic input layer class:

```
        // ...

        template <class _Functype,
                template <class> class... _LayerTypes>
        class base_multilayer_perceptron <_Functype, input_layer, _LayerTypes...>
                : public network <input_layer, _LayerTypes<_Functype>...>
        {// specialized with basic input_layer class

                typedef network_type                    base;

        public:

                // publish typenames...
                typedef _Functype                       function_type;
                typedef input_layer                     input_layer_type;
                typedef perceptron_layer<_Functype>     perceptron_layer_type;
                typedef output_layer<_Functype>         output_layer_type;


                // constructor with a variable number of layer sizes
                template <class... sizes>
                base_multilayer_perceptron(sizes... i)
                        : base(i...) // direct base class initialization
                {
                }

                // destroy
                ~ base_multilayer_perceptron()
                {
                }

                // ...

        private:

                // ...
        };

} // namespace artificial_neural_networks
```

Note that the constructor function is contrived to get a variable number of parameters, herein named *sizes*. Such parameter pack is expanded in the initializer list and used in the base class, in order to build a network endowed with an unknown number of layers, each of which can have a different inner size configuration.

One could discuss whether it is convenient to develop layer classes that have a compile time or run time ability to size their inner vectors. Our choice, for the scope of this research,

was to allow runtime selection of network layers, in order to be able to create differently sized neural networks at runtime having the same fixed number of layers. However, if the implementor should have information about the configuration requirements, he should privilege the compile time decision whenever possible.

In the following last code, a multilayered perceptron class is finally instantiated from a C++ file, and it can be used to build objects of the same type:

```
int main()
{

// ...

typedef self_organizing_map                     map_type;

typedef ann::base_multi_layer_perceptron <ann::hyperbolic_tangent,
        ann::input_layer,
                ann::perceptron_layer,  // I hidden layer
                        ann::perceptron_layer,  //II hidden layer
                                ann::perceptron_layer,  //III  hidden layer
                                        ann::output_layer>
                                                mlp_type;

typedef perceptron_hive <map_type, mlp_type>    perceptron_hive_type;

perceptron_hive_type PH(SOM_GRANULARITY, MLP_GRANULARITY);

// ...

PH.create_mlps(NETWORKS,        // number of hives
                MLP_INPUT_SIZE,
                        MLP_HIDDEN_SIZE,        // I
                                MLP_HIDDEN_SIZE,        // II
                                        2*MLP_HIDDEN_SIZE,      // III
                                                MLP_OUTPUT_SIZE);

hive_launch(MS, ED, PH, TESTBEG, TESTEND);

// ...

return 0;
}
```

In the above listed code, object *PH* constructed from class *perceptron_hive* creates one self organizing layer and a large number of neural networks, using the compile time configuration previously instantiated and a few runtime parameters, the latter not impacting the memory resources usage.

5.2 Parametric Recursion

In this section we are going to describe the use of parametric recursion, useful to build template class member functions that are instantiated at compile time.

Since the use of variadic templates such as the *base_multilayer_perceptron* (that we introduced in the previous section) implies generating classes endowed with a variable number of inner components at compile time, every iteration performed among those inner components can as well be performed statically instead of dynamically (i.e. avoiding runtime *"for"* cycles), thus greatly enhancing code efficiency.

The technique used to create parametric recursive calls is to create a set of different template member functions suitable to the same purpose. The first member function serves as an interface to the first call, the remaining two private member functions are used respectively to continue and stop the recursion.

In the code listing that follows we provide an excerpt of the private member functions used in the *base_multilayer_perceptron* template to perform the feedforward procedure on the artificial neural network. The recursive iteration starts from the first hidden layer and runs until the output layer:

```
template <class _Functype,
        template <class> class... _LayerTypes>
class base_multilayer_perceptron <_Functype, input_layer, _LayerTypes...>
        : public network <input_layer, _LayerTypes<_Functype>...>
{// specialized with basic input_layer class

        // ...

private:

        template <class _inIt>
        void _RSfeed(const _inIt& _Beg, const _inIt& _End)
        {// feedforward procedure
                _input_layer().feed_raw(_Beg, _End);

                _feed<1>(); // forward parametric recursion start ...
        }

        template <size_t I>
        void _feed()
        {
                _layer<I>().feed(_layer<I-1>());

                _feed<I+1>(); // next layer parametric call
        }

        template <>
        void _feed<tuple_size_type::value>() {/*do nothing*/} // stop recursion

        // ...
```

In the next code excerpt we provide a second example of parametric recursion (from the

same *base_multilayer_perceptron* template), this time performed from the outmost layer backward to the first hidden layer. This procedure is suitable for backpropagation algorithms.

Similarly to the previous code, the first member function of the three serves as interface for the first call, the second and third members respectively sustain and stop the parametric recursion:

```
        // ...

        template <class _inIt>
        void _RSbackpropagate(const _inIt& _Act)
        {// learning backpropagation procedure
                _output_layer().
                        back_propagate(_Act, _layer<tuple_size_type::value-2>());

                // backward parametric recursion start...
                _backpropagate<tuple_size_type::value-2>();
        }

        template <size_t I>
        void _backpropagate()
        {
                _layer<I>().back_propagate(_layer<I-1>());

                _backpropagate<I-1>(); // previous layer call
        }

        template <>
        void _backpropagate<0> () {} // stop recursion at input layer

        // ...
};
```

Note that in both examples the compiler will explode the inner template member function into a variable number of class member functions, depending upon the number of hidden layers the multilayer perceptron is instantiated with. For example, let us suppose to instantiate the *base_multilayer_perceptron* template with 7 hidden layers, code will be generated for total 9 member functions.

5.3 Discussion

Variadic template features of C++ 14 programming language enables implementors to design highly flexible models to be instantiated at compile time, enhancing code reusability and scalability.

The parametric recursion technique allows a dramatic reduction of the number of runtime routines, thus contributing to maximize code efficiency.

Finally, the above presented code listings are part of the implementation of back propagation multilayer perceptrons we used in all the predictor systems described in the present thesis.

Chapter 6

Concluding remarks

In this thesis we described some novel predictor systems suitable to the analysis and forecast of streaming datasets. All the proposals were tested on the same large dataset (Bitcoin US Dollar hourly price spot rates), in order to test them as benchmarking methods against each other.

In Chapter 2 we describe our first proposed system, based on a Self Organizing Layer featured by slightly novel features if compared to the classical Self Organizing Maps. We use the SOL as a high precision clustering module, in order to generate a real value forecast of the streaming price. We also employ a group of dedicated artificial neural networks (multilayered perceptrons learning via the backpropagation algorithm) trained to forecast the past raw prediction errors, in order to fine tune the final price forecast. Results show the system can outperform the benchmarking methods, as well as to produce substantial profits when integrated into a simple expert advisor suitable to perform spot currency market operations.

In Chapter 3 we explore the shift variance properties of the Fast Wavelet Transform, in order to integrate the wavelet analysis into a prediction system suitable to forecast streaming time series. In the specific, a fixed size window of the input series is first transformed using the FWT, and the wavelet crystals are stored into a suitable container. The process is repeated at each streaming data insertion. The novel shift variance properties of the FWT (whose theoretical guarantees are provided), dramatically increase the efficiency of the calculations and they allow the retrieval of the majority of the wavelet coefficients, thus reducing the number of neural networks necessary to perform a prediction. We also discover linear relationships among the varying (non shift variance theorem) wavelet coefficients, and such relationships help us to greatly improve the prediction accuracy of the system. Finally, we explain how the predictor can benefit of a virtual switching system aimed at selecting the most suitable type of estimator machine, associated to the series to be forecasted according to its statistical properties. Results show this system outperforms the benchmark methods, including the one presented in Chapter 2.

In Chapter 4 we position with the possibility to design and develop a prediction system composed of a preprocessing module based on the Synchrosqueezed Continuous Wavelet Transform. We recall the fundamental properties of the CWTSST as well as the principal stability issues of the aforementioned reassignment technique. We provide some implementation aspects related to the CWTSST and discuss our position to introduce machine learning objects assisted by the decomposition of the input series into Intrinsic Modes. The contents

of Chapter 4 were recently submitted as contributions to the KDWEB 2016 workshop, and presented in Cagliari in September 2016.

Throughout the research period we always tried to design non parametric systems, in order to predispose them for the eventual extensibility to general purpose algorithms. We followed different scientific approaches to the analysis of streaming datasets, including clustering and self organizing methods up to signal processing theory algorithms. Some of the mentioned models are fairly recent, others are less. In any case, we found that it is still possible to find new research venues based on such classical techniques, and we provide an outlook for further research directions in each chapter.

The majority of the experimental work has been done starting from the second year of the PhD school, after having understood the state of the art relative to the herein mentioned scientific approaches.

The prediction systems were developed using a high performance compilable programming language, conforming to international standards and making use of heavy parametric polymorphism, in order to maximize core component reusability. The efficiency of the implementations allowed us to perform extensive tests on large datasets and, in several cases, it was key to discover (even incidentally) what we considered the most important achievements presented in this thesis.

Chapter 7

Acknowledgments

We would like to thank Eugene Brevdo and Jonathan Lilly for their contribution to determine several theoretical misconceptions relative to the signal reconstruction's accuracy after the Synchrosqueezed Wavelet Transform analysis.

We would like to express gratitude to Phillip James Plauger for the support given in learning advanced features of the C++ programming language, particularly those related to the Standard Template Library.

Finally, we would like to thank Roberto Tonelli - UNICA for reviewing several of our draft manuscripts, and our Professor Michele Marchesi for the great support and comprehension in tutoring all the doctoral activities.

Bibliography

[1] G. S. Atsalakis and K. P. Valavanis. Surveying stock market forecasting techniques part ii: Soft computing methods. *Expert Systems with Applications*, 36:5932–5941, 2009.

[2] E. Avci. Forecasting daily and sessional returns of the ise-100 index with neural network models. *Journal of Dogus University*, 8:128–142, 2007.

[3] N. Baba and M. Kozaki. An intelligent forecasting system of stock price using neural networks. *In Proceedings of the international joint conference on neural networks, Baltimore, MD, USA*, pages 371–377, 1992.

[4] Gregory Beylkin, Ronald Coifman, and Vladimir Rokhlin. Fast wavelet transforms and numerical algorithms i. *Communications on pure and applied mathematics*, 44(2):141–183, 1991.

[5] L. Cao. Support vector machines experts for time series forecasting. *Neurocomputing*, 51:321–339, 2003.

[6] G.H. Chen, S. Nikolov, and D. Shah. A latent source model for nonparametric time series classification. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1088–1096. Curran Associates, Inc., 2013.

[7] H. H. Chu, T. L. Chen, C. H. Cheng, and C. C. Huang. Fuzzy dual-factor time-series for stock index forecasting. *Expert Systems with Applications*, 36:165–171, 2009.

[8] P. Ciaian, M. Rajcaniova, and A. Kancs. The economics of bitcoin price formation. *EERI Research Paper Series*, 2014.

[9] Mac A Cody. The fast wavelet transform: Beyond fourier transforms. *Dr. Dobb's Journal*, 17(4), 1992.

[10] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 1988.

[11] I. Daubechies and J.C. Feaveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, XLV:485–560, 1992.

[12] I. Daubechies, J. Lu, and Hau-Tieng W. Synchrosqueezed wavelet transforms: An empirical mode decomposition-like tool. *Applied and Computational Harmonic Analysis*, 30:243–261, 2011.

[13] I. Daubechies and W. Swelden. Factoring wavelet transforms into lifting steps. *Program for Applied and Computational Mathematics, Princeton University*, 1997.

[14] Ingrid Daubechies and Stephane Maes. A nonlinear squeezing of the continuous wavelet transform based on auditory nerve models. *Wavelets in medicine and biology*, pages 527–546, 1996.

[15] A. Dickey and W. A. Fuller. Distribution of estimators for autoregressive time series with a unit root. *J. of the American Statistical Association*, 74:427–431, 1979.

[16] C. L. Dunis and X. Huang. Forecasting and trading currency volatility: an application of recurrent neural regression and model combination. In C. L. Dunis, J. Laws, and P. Naim, editors, *Applied quantitative methods for trading and investment*, pages 129–160. John Wiley & Sons Ltd, 2003.

[17] C. L. Dunis and M. Williams. Application of advanced regression analysis for trading and investment. In Wiley Finance Series, editor, *Applied quantitative methods for trading and investment*, pages 34–35. John Wiley & Sons Ltd, 2003.

[18] L.V. Fausett. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall international editions. Prentice-Hall, 1994.

[19] I. Francis and K. Sangbae. *An introduction to Wavelet Theory in Finance: A Wavelet Multiscale Approach*. World Scientific, 2012.

[20] D. Garcia, C.J. Tessone, P. Mavrodiev, and N. Perony. The digital traces of bubbles: feedback cycles between socio-economic signals in the bitcoin economy. *Journal of the Royal Society Interface*, 11, 2014.

[21] R. Gencay, F. Selcuk, N. Gradojevic, and B. Whitcher. Asymmetry of information flow between volatilities across time scales. *SSRN*, 2009.

[22] S. H. Hsu, J. J. P. A. Hsieh, T. C. Chih, and K. C. Hsu. A twostage architecture for stock price forecasting by integrating selforganizing map and support vector regression. *Expert Systems with Applications*, 36:7947–7951, 2009.

[23] C. L. Huang and C. Y. Tsai. A hybrid sofm-svr with a filterbased feature selection for stock market forecasting. *Expert Systems with Applications*, 36:1529–1539, 2009.

[24] N. Huang. The empirical mode decomposition and the hilbert spectrum for non-linear and non-stationary time series analysis. *Proc. R. Soc. Lond.*, 454:903–995, 1998.

[25] Shuhaida Ismail, Ani Shabri, and Ruhaidah Samsudin. A hybrid model of self-organizing maps (som) and least square support vector machine (lssvm) for time-series forecasting. *Expert Systems with Applications*, 38(8):10574–10578, 2011.

[26] R. Jammazi. Cross dynamics of oil-stock interactions: a redundant wavelet analysis. *Energy*, 44:750–777, 2012.

[27] I. Kaastra and M. Boyd. Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10:215–236, 1996.

[28] J. Kaminski and P.A. Gloor. Nowcasting the bitcoin market with twitter signals. *CoRR*, abs/1406.7577, 2014.

[29] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[30] T. Kohonen. *Self-organizing maps*. second ed. Springer, 1995.

[31] T. Kohonen. Essentials of the self-organizing map. *Neural Networks*, 37:52–65, 2013.

[32] Teuvo Kohonen. The self-organizing map. *Neurocomputing*, 21(13):1 – 6, 1998.

[33] D. Kondor, M. Posfai, I. Csabai, and G. Vattay. Do the rich get richer? an empirical analysis of the bitcoin transaction network. *PLoS ONE*, 9(2): e86197. doi:10.1371/journal.pone.0086197, 2014.

[34] S.G. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, 11 (7):674–693, 1989.

[35] S.G. Mallat. *A wavelet tour of signal processing, 3rd ed.* Academic Press, 2008.

[36] M. Masih and O. Al T. Alzahrani. Systematic risk and time scales: New evidence from an application of wavelet approach to the emerging gulf stock markets. *International Review of Financial Analysis*, 19(1):10–18, 2010.

[37] P. Masset. Analysis of financial time-series using fourier and wavelet methods. *SSRN*, page http://ssrn.com/abstract=1289420, 2008.

[38] M. Matta, I. Lunesu, and M. Marchesi. The predictor impact of web search media on bitcoin trading volumes. *Information Filtering and Retrieval - DART 2015*, 2015.

[39] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[40] J.B. Ramsey. Wavelets in economics and finance: Past and future. *Studies in Nonlinear Dynamics and Econometrics*, 6(3), 2002.

[41] J.C. Reboredo and M.A. Rivera-Castro. Wavelet-based evidence of the impact of oil prices on stock returns. *International Review of Economics and Finance*, 29:145–176, 2014.

[42] B. Roche and M. Rockinger. Switching regime volatility: an empirical evaluation. In C. L. Dunis, J. Laws, and P. Naim, editors, *Applied quantitative methods for trading and investment*, pages 193–208. John Wiley & Sons Ltd, 2003.

[43] A. Rua and L.C. Nunes. International comovement of stock market returns: A wavelet analysis. *Journal of Empirical Finance*, 16(4):632–639, 2009.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back propagating errors. *Nature*, 323:533–536, 1986.

[45] D. Shah and K. Zhang. Bayesian regression and bitcoin. *arXiv:1410.1231*, 2014.

[46] T. Z. Tan, C. Quek, and N. G. See. Biological brain-inspired genetic complementary learning for stock market and bank failure prediction. *Computational Intelligence*, 23:236–261, 2007.

[47] G. Thakur, E. Brevdo, N.S. Fuckar, and H.T. Wu. The synchrosqueezing algorithm for time-varying spectral analysis: Robustness properties and new paleoclimate applications. *Signal Processing*, 93(5):1079–1094, 2013.

[48] R. Tsaih, Y. Hsu, and C. C. Lai. Forecasting s&p 500 stock index futures with a hybrid ai system. *Decision Support Systems*, 23:161–174, 1998.

[49] P. D. Wasserman. *Neural Computing : theory and practice*. Van Nostrand Reinhold, 1989.

[50] Z. Wu and N.E. Huang. Ensemble empirical mode decomposition: a noise-assisted data analysis method. *Advances in Adaptive Data Analysis*, 1:1–41, 2009.

[51] Xiao-Ping Zhang, Li-Sheng Tian, and Ying-Ning Peng. From the wavelet series to the discrete wavelet transform-the initialization. *IEEE Transactions on signal processing*, 44(1):129–133, 1996.