ENHANCING K-NEAREST NEIGHBORS FORECASTING USING DISCRETE WAVELET TRANSFORM

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

presenta

Sebastián Sánchez Maldonado

Juan José Flores Romero

Director de Tesis

Félix Calderón Solorio

Co-Director de Tesis

Universidad Michoacana de San Nicolás de Hidalgo

MORELIA, MICHOACÁN. AGOSTO 2020

Resumen

El trabajo de esta tesis analiza cómo es posible mejorar el pronóstico de k-Nearest Neighbors con el uso de la descomposición wavelet a través de lifting. El lifting es capaz de representar todas las wavelets estándar, así como las wavelets no lineales o de segunda generación. Esto hace que la descomposición de las señales esté mejor adecuada para las tareas de pronóstico.

Para sacar mayor provecho al lifting, sus coeficientes son evolucionados. Esta decisión fue tomada dado que sólo se han usado wavelets estándar para análisis musical para mejorar el pronóstico. El evolucionar libremente los coeficientes nos permite tener, en teoría, las mejores wavelets de primera y segunda generación a nuestra disposición.

A pesar de que se ha investigado el uso de la transformada wavelet, no se ha hecho mucho en el campo de demanda y consumo de potencia eléctrica. También se presentan resultados que superaron aquellos que fueron obtenidos en investigaciones pasadas para las series de tiempo presentadas en esta tesis. Aún así, la mayor contribución consiste en proponer una wavelet cuasi-óptima para descomponer señales en el campo mencionado, dado que, hasta el día que esta tesis fue escrita, no ha habido intentos de evolucionar wavelets para pronóstico, y la wavelet generalmente usada para tareas de pronóstico es la Daubechies 5, la cual prueba esta tesis que no es óptima, incluso en la familia Daubechies.

Palabras clave— lifting, evolución diferencial, series de tiempo, carga de tiempo corto, eliminación de ruido

Abstract

The work in this thesis analyses how it is possible to enhance k-Nearest Neighbors forecasting with the use of wavelet decomposition through lifting. Lifting is able to represent all the standard wavelets, as well as non-linear or second generation wavelets. This makes that the decomposition of the signals is better suited for forecasting tasks.

In order to get the most out of lifting, its coefficients were evolved. This decision was made because only standard wavelets used for music analysis have been considered to enhance forecasting. Evolving freely the coefficients allows us to have, in theory, the best first and second generation wavelets at our disposal.

While research has been done using the wavelet transform, not so much has been made in the field of electric power demand and consumption. We also present results that outperform those that were obtained in previous research for the time series presented in this thesis. Yet the main contribution comes in proposing a quasi-optimal wavelet for decomposing signals in the aforementioned field, given that, to the day in which this thesis was written, there has been no attempt to evolve wavelets for forecasting, and the wavelet broadly used for forecasting tasks is the Daubechies 5, which this thesis proves that is not the optimal, even in the Daubechies wavelet family.

Keywords— lifting, differential evolution, time series, short-term load, denoising

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

τ	Forecasting lead time.
y_t	Sample t of signal y.
τ	Lead time.
$\hat{y}_t$	Forecast for signal y at sample t.
$\hat{y}_t(t-\tau)$	Forecast for signal y at sample t with lead time τ.
e_t	Forecast error.
$e_t(\tau)$	Forecast error associated with lead time τ.
X	Metric space.
x_i	Values of an i-th element in X.
x	Values in X of a query.
x'	Nearest neighbor for query x.
θ_i	Class or function value associated with x_i.
θ	Class or function value associated with query x.
θ'	Class or function value associated with x'.
$x_{(i)}$	Values of an i-th element in X that has been reordered.
$\theta_{(i)}$	Class or function value associated with $x_{(i)}$.
$d(x_i, x_j)$	Distance between x_i and x_j using metric d.
k	Number of considered neighbors.
$\delta(p, h)$	Kronecker delta function between p and h.
R	Risk or probability error.
$R*$	Bayes error rate.
$H(z)$	Low-pass decomposition filter.
$\tilde{H}(z)$	Low-pass reconstruction filter.
$G(z)$	High-pass decomposition filter.
$\tilde{G}(z)$	High-pass reconstruction filter.
ξ	Filter length.
s	Approximation coefficients.
ρ	Detail coefficients.
l	Decomposition level.
s_l	Approximation coefficients at decomposition level l.
ρ_l	Detail coefficients at decomposition level l.
y_e	Even samples of signal y.
y_o	Odd samples of signal y.
$P(y_e)[n]$	Odd predictor of sample n using y_e.
$U(\rho)[n]$	Even updater of sample n using ρ.
w	Window frame used to search for potential neighbors.

$y \downarrow_2$	Downsampling a signal y by 2.
$y \uparrow_2$	Upsampling a signal y by 2.
$x * y$	Convolution of signals x and y.
w	Window frame for considering neighbors.
v	Scaling factor.
k-NN	k-Nearest Neighbors.
CENACE	National Center for Energy Control.
CWT	Continuous Wavelet Transform.
DWT	Discrete Wavelet Transform.
FWT	Fast Wavelet Transform.
SWT	Stationary Wavelet Transform.
FIR	Finite Impulse Response.
ANNs	Artificial Neural Networks.
DE	Differential Evolution.

Chapter 1

Introduction

Having an accurate representation of the future, in order to make more efficient and better present decisions, has been a topic of interest for researchers and people in the industry. For example, due to the privatization and deregulation of power systems in many countries, electricity has entered the competitive market and it is bought and sold accordingly. This makes forecasting electricity load important and the time series field of interest for this thesis, and with good reason since it was published in 1984 that an increase of 1% in the forecasting error would imply in a £10 million in operating costs per year [Bunn and Farmer, 1985]. This cost increase comes from unnecessary spinning reserve when overestimating future load results, and failure to meet the demand and buying last minute expensive electricity when underestimating.

That is just an example of why developing good forecasting methods is important. However, most of the real-world time series tend to be very complex, making the former task a very complex one. A group of complex series are the ones regarding electricity power demand and consumption. In order to tackle the aforementioned problem, researchers have relied on non-parametric tools such as k-Nearest Neighbors (k-NN) [Flores et al., 2019], and others that are able to capture trend and seasonality [Zhang et al., 1998, Zhang, 2003].

Even though those techniques have improved the overall forecasting results, some time series are complex enough so that forecasts do not yield results within the desired error margin. Those series can be simplified by filtering the diverse frequency components

that make them, forecasting those components, and reverting the process in order to get a forecast of the complex one. This process is desired since it simplifies the overall forecasting task.

Filters that decompose the signals as stated above already exist, one of them is the Continuous Wavelet Transform (CWT) and the Discrete Wavelet Transform (DWT) [Daubechies, 1992]. One of the advantages of the DWT over other types of transforms is that the resulting decomposition signals are in the same domain as the original one, so there is no need for extra steps to make the forecasts. However, the DWT relies on a convolution kernel, named wavelet, that makes shorter or broader the frequency range of the underlying high and low pass filters. Since different domain signals have different properties, a wavelet which makes a good decomposition for a signal, might not do so for another. This is why it is important to find quasi-optimal wavelets for a given signal domain. But, since there are so many of them, most of the researchers resort to known wavelet families that improve results, but are by no means the best ones for the signals being used.

Trying to come up with an analytical formula to define a wavelet family for each known signal domain would be a very difficult and time consuming task. However, while it is difficult to evolve the coefficients that make up the wavelets, the DWT itself can be described as a series of lifting steps whose coefficients can be evolved freely while preserving the properties of the DWT [Sweldens, 1996, Sweldens, 1998].

This thesis uses k-NN to forecast short-term load time series. To aid the process, the wavelet transform serves as a denoiser of the time series, evolving the lifting coefficients that define the wavelet to try to minimize the error as much as possible. The short-term load time series used in this thesis are provided by the National Center for Energy Control (in Spanish Centro Nacional de Control de Energía – CENACE), which operates the Mexican Interconnected Power System (MIPS). Since this thesis uses sensible and very important data, all values will be scaled.

1.1 Related Work

Due to its relevance, as stated in the previous section, forecasting short-term electricity load is not something new. For example, in [Pai and Hong, 2005] support vector machines are used to make the aforementioned task. Artificial Neural Networks (ANNs) are also commonly used, being [Zhang et al., 1998] a review of the role they partake in forecasting, and [Hippert et al., 2001] a particular review of the role in short-term load forecasting. In this thesis, k-NN is used and, while not as popular as the others in this forecasting field, in [Fan et al., 2019] research was made with it.

Due to the properties of the DWT, it has been used for a long time for time series analysis and forecasting, with [Wong et al., 2003] as the first work using it to model the trend and seasonality of the signals. The first work done that tackles the problem of decomposing a signal, predict in its components, and add up the results to get a forecast of the original one is shown in [Conejo et al., 2005]. While these are some of the earlier works, to this date there is still research being made on working with time series and DWT. To cite some examples: work has been done comparing different forecasting methods with the aid of the DWT [Stolojescu et al., 2010], decomposing using the DWT and a subsequent forecasting of the components by ANNs to predict monthly water table depth [Anandakumar et al., 2019], and a mixture between mutual information, DWT, evolutionary particle swarm optimization and adaptive neuro-fuzzy inference to predict wind power and electricity market prices [Osório et al., 2014].

Although there is a fair amount of research involving time series forecasting and evolution, this thesis only focuses on evolving wavelet coefficients. Before trying to fully evolve wavelets, adapting wavelets was studied, trying to accomplish this through dictionary methods, where a basis is selected from a set of predefined functions called atoms. Some examples are the best basis algorithm [Coifman and Wickerhauser, 1992] and wavelet packets [Wickerhauser, 1994]. Evolutionary algorithms are used for adaptive dictionary methods in [Lankhorst and Laan, 1995] and [Liu and Wechsler, 2000]. Work using lifting to adapt wavelets by optimizing data-based prediction error criteria is done in [Claypoole et al., 1998]. [Erba et al., 2001] and [Lee et al., 1999] presented work regarding evolution

of digital filters in general, not only wavelets. Stochastic optimization techniques have also been used, like in [Monro and Sherlock, 1997] where minimization through simulated annealing is used to design wavelets with balanced space and frequency dispersions, and in [Hill et al., 2001] the genetic algorithm is used to find trigonometric functions that define a CWT.

[Vaithiyanathan et al., 2014] attempted to evolve wavelet coefficients. The only successful approach was to search for wavelets in the space near the coefficients of the CDF 9/7 wavelet, used as standard for image compression, by adding low amplitude Gaussian noise. To this date, the only work that successfully evolves wavelets through lifting can be found in [Grasemann and Miikkulainen, 2004], where Grasemann et al. use a coevolutionary genetic algorithm to find lifting coefficients that define a quasi-optimal wavelet for cubic spline compression. Then, in [Grasemann and Miikkulainen, 2005], Grasemann et al. apply the previous methodology to find lifting coefficients that define a wavelet that outperforms the FBI wavelet in terms of fingerprint compression, being that the FBI wavelet was used a standard since it was considered the best for the task [Hopper et al., 1993]. To the best of our knowledge, there is no documented attempt to evolve wavelet coefficients to forecast time series of any nature.

1.2 Problem Statement

Given an electricity load time series and k-NN as its predictor, determine the lifting coefficients that define a quasi-optimal wavelet for forecasting purposes.

1.3 Hypothesis

It is possible to use the wavelet transform as a denoiser to aid the forecasting of short-term load time series, using k-NN as the predictor, improving its results. In particular, it is possible to find a wavelet through evolution of lifting coefficients that outperforms the forecasting that is achieved by using wavelets from standard wavelet families.

1.4 Thesis Objectives

1.4.1 General goal

Define a lifting scheme with enough wavelet representation capacity, getting its coefficients through evolution that result in a signal decomposition that, along with an k-NN, outperforms the forecasting done with or without the aid of the DWT. When compared to a forecasting task aided by the DWT, wavelets used in most of the research to date will be considered.

1.4.2 Particular goals

1. Define a scheme that combines k-NN with wavelets, varying the level of decomposition and k-NN hyper-parameters, until the best results using that scheme are found for electricity load time series.
2. Search for the wavelet that yields the best results.
3. Successfully implement a lifting scheme that decomposes signals just like the DWT.
4. Finding the lifting steps required to have enough capacity for the desired decomposition.
5. Use evolution to find the lifting coefficients that outperform the results obtained in the second goal and those already provided in previous research.

1.5 Motivation

Even though there is some work done in terms of electricity load forecasting, taking into account the total of researchers in computer science, only a small percentage of them is interested in forecasting, compared to more popular fields of research. In many countries, such as Mexico, only a handful of people know how to do forecasting and the results obtained so far in the industry can be improved, which opens work opportunities. This improvement of results is what motivated the search for new forecasting schemes, resulting in the one proposed.

Another motivation comes from seeing how much enhancement can be made to forecasting schemes using evolved wavelets. Since nobody has evolved wavelets using lifting to forecast, this work may help to further explore this field of research.

1.6 Justification

Most of the forecasting done involving wavelets use the Daubechies family, in particular the order 5 wavelet, which corresponds to the coefficients that define the filter used to decompose the time series. While the obtained results are satisfactory, no research has been done to determine if that wavelet is the best suited in that family to electricity load forecasting. Even if that was the case, [Grasemann and Miikkulainen, 2005] proves that the likelihood of finding a better wavelet through lifting and evolution is high.

Finding a quasi-optimal wavelet for electricity load forecasting would improve the results for anyone attempting to perform this task, even if they are not using k-NN to make the predictions or if data from another region is being used, since decomposing using lifting is simple and inexpensive in terms of time.

1.7 Thesis Layout

Chapter 2 presents a general background of all the methods and schemes used in the work of this thesis. We explain concepts of time series analysis and forecasting. We give a brief introduction to k-NN. We discuss definitions of some types of wavelet transforms. Finally, we present an explanation of differential evolution. The proposed method and tests used in order to obtain the lifting coefficients in this thesis, along with its explanation, can be found in Chapter 3. The forecasting results obtained with the found lifting coefficients, along with a comparison with forecasting without the use of wavelets and with the use of the Daubechies family, is found in Chapter 4. Finally, in Chapter 5 a summary of the results along with a personal interpretation is given. This chapter also includes recommendations for future work regarding the subject.

Chapter 2

Preliminaries

In order to fully explain the work done and presented in this thesis, some general background needs to be explained in this chapter. It is assumed that the reader is familiar with basic concepts of linear algebra, calculus, probability and statistics.

This chapter shows some general concepts of time series analysis and forecasting. It also features an explanation of k-Nearest Neighbors, since it is used as the predictor. It also explains the wavelet transform, along with the lifting implementation of it, since we use it a a denoiser to enhance the forecasting done by the predictor. Finally, it explains differential evolution, which we use to try to find the lifting coefficients that perform the best denoising of the time series.

Before we begin explaining the concepts of this chapter, we define the notation that is used. When preceded by a symbol, values enclosed in parenthesis () denote the arguments for the function associated with the symbol, and values enclosed in brackets [] denote a specific sample of the array associated with the symbol. We use subscript $_i$ to denote the i-th element of the indexed variable that precedes it. Aside from this, we use the standard notation found in the bibliography of this thesis.

2.1 Time Series Analysis and Forecasting

This section discuses some basic concepts regarding time series and forecasting that are fundamental to understand this thesis. Should the reader want to delve more in

the subject, it is advised to refer to [Montgomery et al., 2015], since that is the material that was consulted for the writing of this section.

First, it is important to define some concepts that will be used throughout this thesis. The forecasting problems used in this thesis involve time series, which are time-oriented or chronological sequence of observations on a variable of interest; the values of that variable are typically collected at equally space time periods. We will use the term forecast to refer to a prediction of some future event or events. Forecasting problems can be classified depending how far in the future the predictions take place; for this thesis we focus only on short-term forecasting, which involves predicting events days, weeks, or months into the future. To make the forecasts, forecasting models are used, which extrapolates past and current behavior into the future. Before looking at concepts that require symbolic notation, forecast horizon and forecast interval shall be defined. The forecast horizon or forecast lead time refers to how far into the future the forecasts will be produced. The forecast interval tells how frequently the new forecasts are prepared. For example, in the time series used in this thesis, in order to test the performance of the predictor, CENACE requires a prediction of the values for a whole day (lead time). The dispatch intervals are 15 minutes long (forecast interval), giving a total of 96 predictions.

Introducing symbolic notation to provide a formal definition, regarding time series, we assume that there are T samples of data available, that go from the first or period 1 to the last or sample T. Let y be the vector that contains the values of a time series, we will use y_t to denote the observation of this variable at time period t, $t = 1, 2, \ldots, T$. The variable can represent a cumulative or an instantaneous quantity. When forecasting, it is important to make a differentiation between the real value at y_t and the predicted one. Taking τ as the forecast lead time, the forecast for y_t at time period $t - \tau$ will be denoted by $\hat{y}_t(t-\tau)$. A common type of forecast is the one step ahead, which predicts y_t one period prior and is represented by $\hat{y}_t(t-1)$. In general, the fitted or predicted value of y_t will be denoted by $\hat{y}_t$. For the CENACE time series, due to the time the operators take to execute the necessary actions, forecasts two steps ahead must be made, denoted by $\hat{y}_t(t-2)$.

When predicting the value of a variable, often being a real number, the probability for the prediction to be equal to the observation tends to 0, meaning that the forecast error

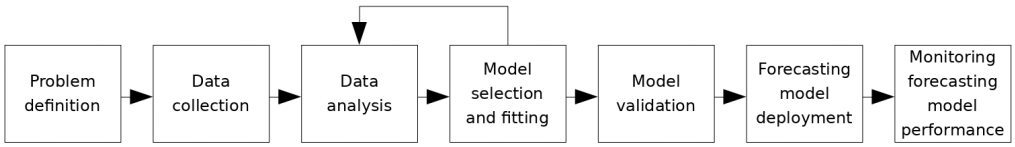

Figure 2.1: Forecasting process.

will never be zero. More specifically, the forecast error that results from a forecast of y_t that was made at a time period $t-\tau$ is called the lead $-\tau$ forecast error and it is represented by

$$e_t(\tau) = y_t - \hat{y}_t(t-\tau). \tag{2.1}$$

To calculate the forecasting error of a one step ahead prediction, the next equation would be used

$$e_t(1) = y_t - \hat{y}_t(t-1).$$

In order to finish this section, a brief explanation of the steps involving a forecasting process will be given. In Figure 2.1 we show a diagram that depicts those forecasting steps and the interaction they have between themselves.

1. Problem definition corresponds to the part of the process where it needs to be understood how the forecasts will be used by the final user. Here is where the forecast lead time and the forecast interval are defined. Also the final user will state the required level of forecast accuracy.

2. Data collection refers to acquire the relevant history for the variable of interest. In most real life scenarios, the data collected is entirely made of normal measurements; frequently, missing values or outliers will be found, along with data problems that may be presented due to the nature of the underlying system or changes in it. One needs to deal with this type of problems, since only representative data of the current underlying system needs to be used.

3. In the data analysis step, plottings of the time series are made in order to detect visually some patterns of it. While observing the plot, one should search for trends and seasonal or other types of cyclical components. We will use the term trend to

indicate an evolutionary movement, whether upward or downward, in the values of the variable. Trends can be both long or short-term. Seasonality refers to behavior that repeats along time in the variable. This behavior can be daily, weekly or even yearly, depending on the process being examined.

4. Model selection and fitting, as the name suggests, consists in choosing a model or models and fitting those to the data. Fitting refers to estimating the model parameters unbeknown to us.

5. In the model validation step, the model is evaluated in a simulation of a real use of it, with data not available in the fitting process. Since it is important to know how the model will behave when used by the end user and there is a big chance that the performance in the fitting step is better than the one with real values, the data is split into a training set and a validation set. The training set will be destined for the model fitting, and the validation set will be used to simulate how the model will perform with new data.

6. During the forecasting model deployment step, the end user will use the model and obtain the desired forecasts. In this part, model maintenance may be given in order to extend the longevity of the model.

7. When monitoring forecasting model performance, the model will be checked regularly to ensure that its performance is still acceptable. Since it is possible and likely to have changes in the properties of the underlying process being predicted, the models may deteriorate and not be useful any longer. If this is the case, it may be necessary to go back to the data analysis step. A constant measurement of the error is a good practice to check the model performance through time.

2.2 Nearest Neighbors

This section explains the k-NN algorithm. Although there are different approaches when it comes to implementing k-NN, this thesis will only focus in its original implementation. Although k-NN was originally conceived as a classification algorithm, it is also possible

to use it for regression; we also show in this section an explanation of how to forecast using it. Most of the concepts are taken from [Kantz and Schreiber, 2004], and we advise the reader to consult it if they want to delve more in the subject.

The inception of k-NN comes from the necessity to perform non-parametric classification, since there are situations when the parametric estimates of probability densities are unknown. Originally introduced in [Fix, 1951, Fix and Hodges Jr, 1952] and further more studied with a more formal approach in [Cover and Hart, 1967], the k-NN algorithm has its roots on what is known as the nearest neighbor rule (NN rule).

2.2.1 NN Rule

Let X be a metric space. Let d be a metric defined in X. We define a set of n pairs $(x_1, \theta_1), (x_2, \theta_2), \dots (x_n, \theta_n)$, being the x_i's values in X, and the θ_i's values in $[1, 2, \dots, M]$, corresponding to the indexes of the different categories that can be assigned to an individual. We will say that the ith individual belongs to the category associated by index θ_i, and x_i are the measurements taken for that individual.

When a new x or query is given, our goal is to find the θ that forms the pair (x, θ). Since we can only observe the measurement x, we will use the information of the already known x_i's and θ_i's to define θ. We will define the nearest neighbor $x' \in \{x_1, x_2, \dots, x_n\}$ to x if it satisfies

$$x' = \underset{i=1,2,\dots,n}{\arg\min}\,(d(x_i, x)).$$

The NN rule decides that x belongs to the category given by θ', corresponding to its nearest neighbor x', and excluding any information that could be provided by the $n-1$ remaining individuals. When $\theta \neq \theta'$, it is considered as a mistake, defining the risk or probability of error R as the ratio of the amount of errors to the number of classifications.

In order to determine the nearest neighbor, it is possible to use one of several known metrics, such as:

- Euclidean: $d(x_i, x) = \sqrt{(x_i - x)^2}$.
- Euclidean squared: $d(x_i, x) = (x_i - x)^2$.

- Manhattan: $d(x_i, x) = |x_i - x|$.

The most common of them is the Euclidean and it is also the one suggested for the purpose of this thesis [Kantz and Schreiber, 2004].

The advantage of this algorithm is that there is no training or fitting process in which values are adapted to represent the underlying distribution for the individuals. The disadvantage, however, is that each time a new query is presented, its distance with every other individual is calculated in order to apply the rule. This makes that the time required to output the result once the query is presented is greater than the time required with many other classification algorithms.

k-NN Rule

It is possible to extend the definition of the NN rule in order to consider more than just one individual; when fixing the number of individuals that are going to be considered to k, the rule is known as the k-NN rule. Given the set of n pairs $(x_1, \theta_1), (x_2, \theta_2), \dots (x_n, \theta_n)$ and the x, we reorder the pairs in the form of $(x_{(1)}, \theta_{(1)}), (x_{(2)}, \theta_{(2)}), \dots (x_{(n)}, \theta_{(n)})$ such that

$$d(x_{(1)}, x) \leq d(x_{(2)}, x) \leq \cdots \leq d(x_{(n)}, x).$$

Then we take the first k pairs $(x_{(1)}, \theta_{(1)}), (x_{(2)}, \theta_{(2)}), \dots (x_{(k)}, \theta_{(k)})$ in order to find the most concurrent category among the $\theta_{(i)}$'s by counting. To formally denote this counting, we will use the Kronecker delta function, which compares two inputs, outputting 1 when they are equal, and 0 otherwise

$$\delta(p, h) = \begin{cases} 0 \text{ if } p \neq h, \\ 1 \text{ if } p = h. \end{cases} \tag{2.2}$$

Using equation (2.2) we can now define a function $C(S, j)$ that counts the number of occurrences of an element j over a sequence S of length m as

$$C(S, h) = \sum_{i=1}^{m} \delta(h, S_i) \tag{2.3}$$

Finally, we use equation (2.3) to define a function that counts the votes for each possible category of the k selected neighbors S_k and outputs the one with the most votes

$$\theta' = \arg\max_i (C(S_k, \theta_{(i)})).$$

The k-NN rule decides that x will belong to the category given by θ', corresponding to the one with the most occurrences among those that were given by the k selected individuals, ignoring the information provided by the $n - k$ remaining.

Although there are techniques that help in the selection of k in order to achieve the minimal error [Kantz and Schreiber, 2004], the optimal value is not necessarily reached by those techniques and varies depending on the distribution in which the individuals are taken. However, it is proved in [Cover and Hart, 1967] that the risk R has bounds

$$R^* \leq R \leq R^* \left(2 - \frac{M}{M-1} R^*\right). \tag{2.4}$$

In equation (2.4), R^* corresponds to the Bayes error rate, which is the minimal error rate possible for a classifier.

2.2.2 k-NN Regression

In order to forecast using k-NN, we need to change its purpose from classification to regression; that is changing the approach of the algorithm from assigning a category to a query to define the behavior of an underlying function at query point x [Kantz and Schreiber, 2004].

Using notation from Section 2.1, given the time series y, we are going to define a database of individuals that are going to be used for the query. We will introduce parameter m which will define the length of the vectors that make the individuals. These individuals are equivalent to the vectors of measurements used for classification. Each individual x_i will start from observation y_i, having the values of the next $m - 1$ observations as well. This can be expressed as

$$x_i = [y_i, y_{i+1}, \ldots, y_{i+m-1}]. \tag{2.5}$$

For time series, instead of categories, we will now use θ_i, corresponding to a function of x_i for the general case, to represent the observation ahead of the last in x_i by lead time τ,

represented as

$$\theta_i = y_{i+m-1+\tau}. \tag{2.6}$$

There will be an individual for each unique vector constructed from y using equations (2.5) and (2.6), and as long as there an observation at the required indexes.

Once the individuals database has been created and query x arrives, which usually corresponds to the vector immediately after the last in the database, we will proceed exactly like we did with the k-NN rule for classification, however we will make a modification to define θ'. We now define $\theta' \in \mathbb{R}$, since it corresponds to the forecast at $\hat{y}_{T+1}$, instead of being an identifier for a category.

Instead of having a vote for each of the k considered neighbors, we will average their associated $\theta_{(i)}$ value, such as

$$\theta' = \frac{1}{k} \sum_{i=1}^{k} \theta_{(i)}.$$

In regression, the probability of $\theta = \theta'$ is 0, so we will define error using equation (2.1).

It is possible to construct the database and calculate θ' in various ways by incorporating more parameters [Kantz and Schreiber, 2004]. However, this thesis uses the procedure explained above since it is simpler, requires a shorter search for the ideal parameter values, and was the one that exhibited the best results while testing.

2.3 Wavelet Transform

Although there is a significant amount of theoretical background regarding the wavelet transform, this section will provide a light version of it that is enough for the reader to understand the role it plays in this thesis. Most concepts will be taken from [Daubechies, 1992], and it is advised for the reader to use that material if a more in depth explanation is desired.

We will particularly focus our attention on the *Discrete Wavelet Transform* (DWT). However, being discrete or continuous, one can think of the wavelet transform as a tool that splits a signal into different frequency components, representing each of those components

with a resolution associated to its scale.

2.3.1 Discrete Wavelet Transform

The DWT serves to represent a signal in terms of translations and dilatations of a function ψ, known as the mother wavelet. This transform will create a set of low resolution coefficients, known as approximations, and a set of finer resolution coefficients, known as details. Rather than deriving and using a formula based on function ψ, an algorithm known as *Fast Wavelet Transform* (FWT) is used to compute the aforementioned coefficients, recursively applying a pair of digital filters to the signal.

Without delving too much on the subject, since it is not a main concern in this thesis, we will define a digital filter as a sequence of real numbers called filter coefficients. Those filter coefficients are convoluted with the desired signal to be filtered. The type of filters used for the FWT algorithm are known as *Finite Impulse Response* (FIR), which means that, when applied to a unitary impulse, the resulting coefficients are non-zero only on a finite range.

The pair of filters used for decomposition consists of a low-pass filter $H(z)$ and a high pass filter $G(z)$. To synthesize the original signal, an inverse transform is applied using low-pass filter $\tilde{H}(z)$ and high-pass filter $\tilde{G}(z)$. The filters $H(z)$, $G(z)$, $\tilde{H}(z)$ and $\tilde{G}(z)$ must form a biorthogonal basis in order to have a perfect reconstruction [Daubechies, 1992]. From now on we are going to treat biorthogonality and invertible as synonyms in a wavelet context. There are some special cases where the corresponding decomposition filters and synthesizing filters have the same values, but mirrored, forming an orthonormal system.

The previous explanation can be expressed in the following equation, considering a signal y and filter length ξ

$$s[n] = \sum_{i=1}^{\xi} y[n-i]H[i] \tag{2.7}$$

$$\rho[n] = \sum_{i=1}^{\xi} y[n-i]G[i]. \tag{2.8}$$

For the sake of completeness, we point out that s in equation (2.7) corresponds to the approximation coefficients and ρ in equation (2.8) correspond to the detail coefficients.

Since we have less frequency components, it is possible to have half the amount of values in s and d compared to y while having enough information to capture the whole behavior of it. Denoting a downsampling by 2 as $y \downarrow_2$, it is possible to write both equations (2.7) and (2.8) as

$$s[n] = \left(\sum_{i=1}^{\xi} y[n-i]H[i] \right) \downarrow_2 \tag{2.9}$$

$$\rho[n] = \left(\sum_{i=1}^{\xi} y[n-i]G[i] \right) \downarrow_2 . \tag{2.10}$$

The inverse transform is obtained in a similar fashion as with the decomposition in the sense that an upsampling is done to the approximation and detail coefficients, followed by a convolution with their respective filters, and finally adding both of their results to obtain the original signal. This can be expressed in the following equation, which uses a simplified notation, denoting $\uparrow_2$ as upsampling by 2 and $*$ as the convolution, as in equation 2.11.

$$y[n] = (s[n]) \uparrow_2 *\tilde{H}[n] + (\rho[n]) \uparrow_2 *\tilde{G}[n]. \tag{2.11}$$

One of the properties of the DWT is that one can further decompose the remaining frequency components by iterating on the output of the low-pass filter. The amount of iterations applied is known as the decomposition level. Since each decomposition adds another array of detail coefficients, for a given decomposition level l, we are going to end up with l arrays of detail coefficients and one with approximation coefficients at the last level. We can use a numeric subscript to denote the corresponding level of decomposition for the coefficients, and if we consider $s_0 = y$, we can write equations (2.9) and (2.10) in a way that generalizes the concept of a multilevel decomposition as

$$s_l[n] = \left(\sum_{i=1}^{\xi} s_{l-1}[n-i]H[i] \right) \downarrow_2$$

$$\rho_l[n] = \left(\sum_{i=1}^{\xi} s_{l-1}[n-i]G[i] \right) \downarrow_2 .$$

Similarly, we can write equation (2.11) to have the same consideration as

$$s_{l-1}[n] = (s_l[n]) \uparrow_2 *\tilde{H}[n] + (\rho_l[n]) \uparrow_2 *\tilde{G}[n].$$

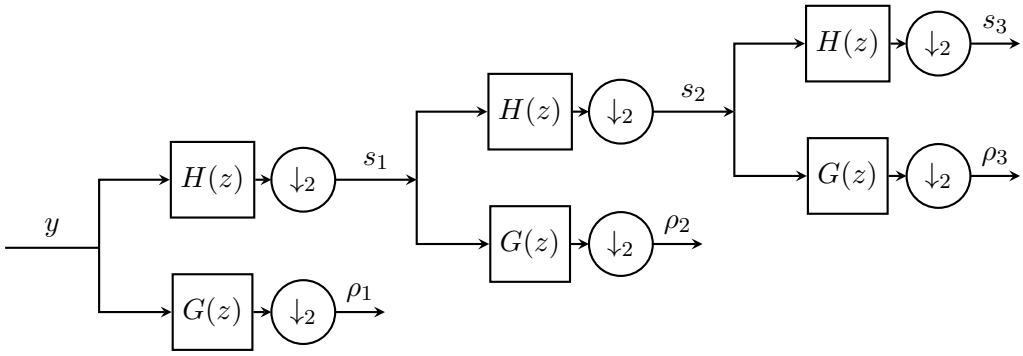

Figure 2.2: A level 3 DWT decomposition using filters.

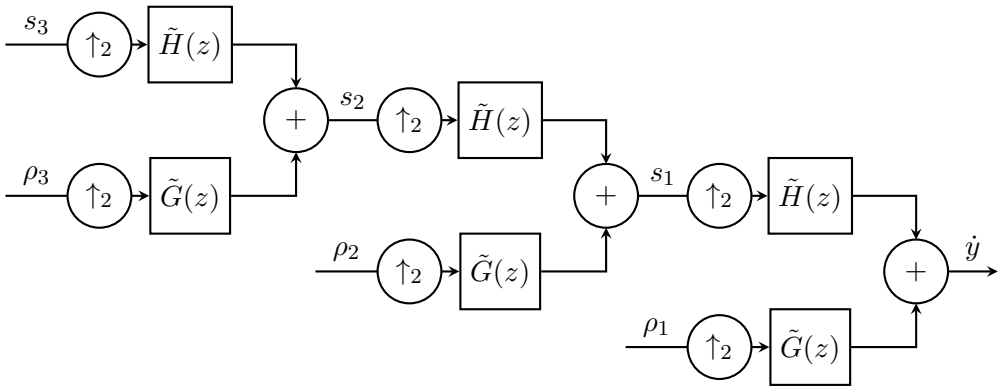

Figure 2.3: A level 3 DWT reconstruction using filters.

In order to show in a clearer way the multilevel properties of the DWT, a diagram is shown in Figure 2.2. The result of the decomposition will be $l+1 = 4$ signals or coefficient arrays ρ_1, ρ_2, ρ_3 and s_3. The higher frequency components will be located in ρ_1 and the lower ones in s_3. It is important to remember that, if y is of length T, then ρ_1, ρ_2, ρ_3, and s_3 will have a length of $T/2$, $T/4$, $T/8$ and $T/16$, respectively.

For the sake of completeness, we show in Figure 2.3 a diagram of the reconstruction process when the inverse wavelet transform is applied through the use of digital filters. It is worth nothing that the output of the reconstruction is denoted by $\dot{y}$ since, even though the signal coefficients are perfectly reconstructed, there is a shift in the reconstructed signal that varies depending on the length of the filter coefficients.

2.3.2 Stationary Wavelet Transform

While we described how we could apply the wavelet transform to time series in theory, and even if what we described is used in almost any type of digital signal, the coefficient decimation poses a problem. First, considering a decomposition level l, the time series length must be divisible by 2^l; one can think of padding or truncating the series at the beginning as a workaround. However, when we make the predictions, the total number of predictions must also satisfy the divisibility condition, but it needs to do so in each of the coefficient arrays. For example, if we wanted to make 16 predictions on the original series, for a level 3 deconstruction, we would be required to make 2 predictions for s_3 and ρ_3, 4 for ρ_2, and 8 for ρ_1, in order to be able to reconstruct the coefficients in terms of the original signal. If we want to make m number of total predictions such that $m \not\equiv 0 \mod 2^l$, then we must use a number $n < 2^l$ that satisfies $m + n \equiv 0 \mod 2^l$, and make a total of $m + n$ predictions. This can be detrimental for the quality of the predictions. Trying to pad the predictions in the reconstruction phase would pose an even worse outcome than the previous solution attempt, since we would be modifying the frequency components that define the behavior of the original signal.

The situation presented in the previous paragraph can and is avoided with the *Stationary Wavelet Transform* (SWT). The SWT is defined in [Shensa, 1992] and averages over all possible shifts of the input signal, the reason being that it is also known as shift-invariant. This is done by removing the time-varying decimators, which means that the coefficient arrays no longer reduce their length with respect to the original. So, for a decomposition level l and a signal of length T, we will end up with lT coefficients, instead of T using the DWT. Subsequent iterations at a given level l must use the original versions of the wavelet filters, expanded by 2^l. The aforementioned process is shown as a diagram in Figure 2.4, considering a level 3 decomposition. Reconstruction is omitted, since it is easy to see the similarity and slight differences with the DWT.

This type of wavelet transform is also known as redundant since, as stated in Subsection 2.3.1, we only need half the coefficients per level to capture the signal behavior and we have more in the SWT. However, since the wavelet transform takes its input in the

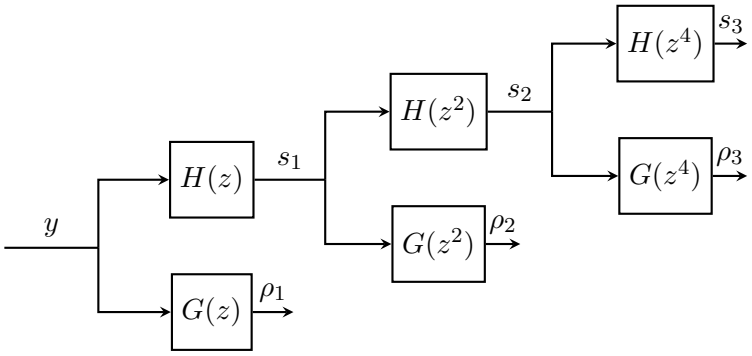

Figure 2.4: A level 3 SWT decomposition using filters.

time domain and its output is also in the time domain, we can perform forecasting on the various components as if it were the case of doing so on the original signal. We can even just predict the lower frequency components and have the results as the overall predictions for the series, which corresponds to using the SWT as a denoiser.

2.4 Lifting

Lifting was conceived as a way to apply non-linear wavelet transforms, also known as second generation wavelets [Sweldens, 1996, Sweldens, 1998]. Lifting provides an entirely spatial-domain interpretation of the transform, which allows to use the transform in signals with more complex geometry and irregular sampling. This introduces nonlinearities while retaining control of the multi-scale properties, hence the name. In [Sweldens, 1996, Daubechies and Sweldens, 1998] it is also proven that any first generation wavelet, as the ones described in Section 2.3, can be represented through lifting. It is important to bear in mind that not every decomposition done with lifting has a first generation wavelet representation.

The lifting scheme is composed by 3 steps. These steps come from viewing the DWT as a prediction-error decomposition. The coefficients that define the wavelet, at a given decomposition level l, serve as predictors for the values at the next level $l + 1$. The detail coefficients are considered prediction errors between the wavelet coefficients and the

values of the next level they are attempting to predict.

The lifting steps are defined as:

- Split: Divide the original signal into two disjoint sets. Since most digital signals come from a smooth and slowly varying underlying function, it is possible to consider that the odd and even samples are highly correlated. If we take this notion to a local context, then it is possible to predict the odd samples from the adjacent even ones. This means that we can perform what is known as the *Lazy Wavelet Transform* (LWT) [Daubechies and Sweldens, 1998], consisting in, given the original signal $y[n]$, create two subsets $y_e[n] = y[2n]$ and $y_o = y[2n+1]$, which corresponds to separating the signal into two that contain the even samples and the odd samples, respectively.

- Predict: We predict the odd coefficients $y_o[n]$ from the even coefficients y_e that neighbors them, using interpolation. This means that the predictor for each $y_o[n]$ is expressed as a linear combination, using scalar α, as

$$P(y_e)[n] = \alpha(y_e[n-1] + y_e[n]).$$

  Given the predictor function, we can now define the first lifting step as the prediction error

$$\rho[n] = y_o[n] - P(y_e)[n]. \tag{2.12}$$

  Going by the same notation as in Section 2.3, this first lifting step is the equivalent to applying a high-pass filter to the input signal. Assuming that the underlying function is smooth, the values in $\rho[n]$ are going to be small. This step is totally reversible by doing

$$y_o[n] = \rho[n] + P(y_e)[n]. \tag{2.13}$$

- Update: We transform the even coefficients $y_e[n]$ into the result of passing the original signal through a low-pass filter and downsampling it. For this, we proceed in a similar fashion as we did in the second step, using a linear combination of the prediction errors

$\rho[n]$

$$U(\rho)[n] = \beta(\rho[n] + \rho[n+1]).$$

We replace the even values $y_e[n]$ by

$$s[n] = y_e[n] + U(\rho)[n].$$

Just as with equation (2.12), given the approximation coefficients $s[n]$ and the detail coefficients $\rho[n]$, it is possible to reconstruct perfectly $y_e[n]$ with

$$y_e[n] = s[n] - U(\rho)[n].$$

If we further use equation (2.13), then we also recover $y_o[n]$, being able to reconstruct the original input signal $y[n]$, making the whole process invertible.

We can repeat as much as we need the second and third step, remembering that $s[n]$ replaces $y_e[n]$ and $\rho[n]$ replaces $y_o[n]$, and using scalars that are not necessarily equal to α or β for this extra steps. This repetition of steps extends our capacity to represent wavelets, whether they belong to the first or second generation. When we are done with the lifting steps, we multiply the final approximation coefficients by a scalar ζ and the detail coefficients by $1/\zeta$. This is done to normalize the energy of the underlying scaling and wavelet functions. In Figure 2.5 we show a diagram depicting the whole lifting scheme, without repeating the predict and update steps pair, showing in Figure 2.6 a diagram of the inverse process.

Just as with the DWT, it is possible to apply multilevel lifting iterating over the low-pass coefficients, so Figure 2.2 already serves the purpose of illustrating the process through a diagram.

In [Daubechies and Sweldens, 1998], along with others, some properties of lifting relevant to this thesis are mentioned:

- Lifting preserves biorthogonality.
- Any wavelet can be expressed as a sequence of lifting steps.

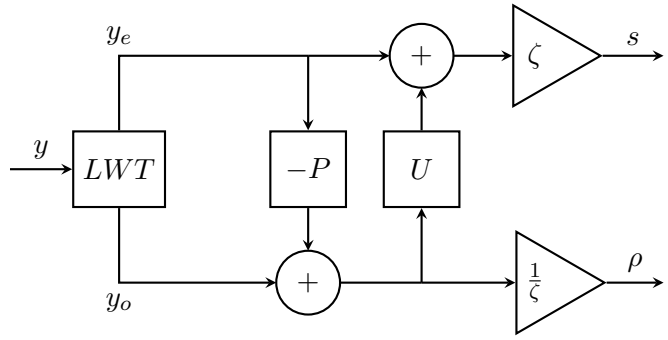

Figure 2.5: Applying lifting steps to a signal.

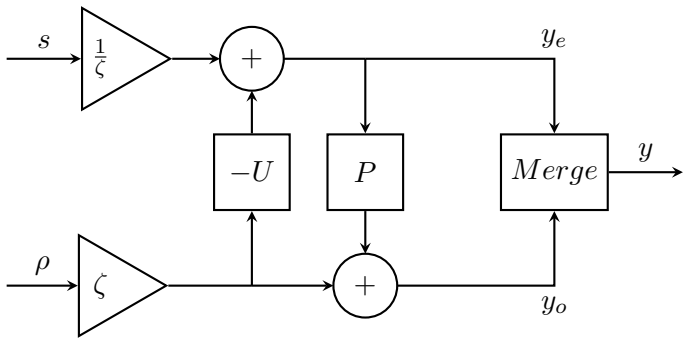

Figure 2.6: Applying inverse lifting steps to reconstruct a signal.

- Lifting is faster than the standard FWT implementation.
- It is easy to implement non-linear wavelet transforms using lifting.
- Lifting is not a unique process. It is possible to have groups of different lifting coefficients that decompose a given signal exactly like the others.

2.4.1 Redundant Lifting

Just as with the DWT, the lifting scheme halves the total number of signal samples at each decomposition level. Thankfully, although for adaptive purposes, a redundant lifting is defined in [Claypoole et al., 1998].

The redundant lifting consists simply in intertwining the output of two non-redundant lifted wavelet transforms. The first transform is exactly as the one defined previously, that is, predict the odd coefficients from the even coefficients. The second transform has the same steps, but predicts the even coefficients from the odd coefficients, achieved by shifting the input by one. The output will be two signals from the same length as the one used as input. The low-pass even coefficients $s_e[n]$ come from the low-pass coefficients of the normal lifting and the low-pass odd $s_o[n]$ coefficients come from the low-pass coefficients of the shifted lifting. The process to obtain the high-pass coefficients is analogous to the previous, but taking the high-pass coefficients. To make this process clearer, in Figure 2.7 a diagram depicting the redundant lifting scheme is presented, where the $Lift$ block represents the regular lifting process shown in Figure 2.5.

Given $s[n]$ and $\rho[n]$, it is possible to reconstruct the original signal by splitting both of them in their respective even and odd coefficients. Given one of the pairs $(s_e[n], \rho_e[n])$ or $(s_o[n], \rho_o[n])$, it is possible to perform the reconstruction following the process showed in Figure 2.6. From Figure 2.7, it should be obvious to the reader that we only need to reconstruct using one pair; using both would imply doing unnecessary extra work.

Just as with regular lifting, it is possible to apply extra lifting steps. It is also possible to perform a multilevel deconstruction, however, as noted with lifting, there is no necessity to extend the lifting coefficients responsible for the decomposition.

The previous descriptions of k-NN and lifting define the tools that are used to

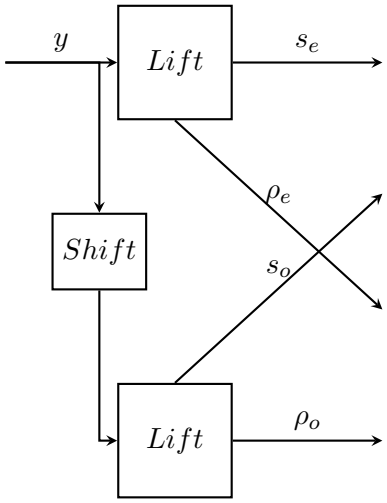

Figure 2.7: Applying redundant lifting to a signal.

make the predictions, however we do not use the lifting representation of a standard wavelet. Instead we evolve the lifting coefficients to find a wavelet transform that yields better results than those obtained with standard ones. The evolution algorithm that this thesis uses is defined in the next section.

2.5 Differential Evolution

Differential Evolution (DE) is a metaheuristic that optimizes functions, defined in [Storn and Price, 1997]. It attempts to find the global minimum by evaluating a population that changes from generation to generation. For the population at any given generation G, it utilizes N_p parameter vectors of dimension D

$$\nu_{i,G} \quad i = 1, 2, \ldots, N_p.$$

The number of individuals N_p remains constant during the optimization process. The initial population vectors are chosen randomly, trying to cover the entire search space. New parameter vectors are generated at every generation by an operation called mutation, consisting in the addition of the weighted difference between two population vectors to a

third vector. The mutated vector's parameters go through what is known as crossover, which consists in mixing them with another predetermined vector's parameters. If the resulting vector from the mutation and crossover yields a lower cost function value than the target vector used at the beginning of this process, then it replaces it in the next generation, being this operation known as selection. Each population vector has to serve once as the target vector so that N_p competitions take place at every generation.

More formally, each operation of DE can be described in the following manner:

- Mutation: For each target vector $\nu_{i,G}$, a mutant vector $\upsilon_{i,G+1}$ is generated

$$\upsilon_{i,G+1} = \nu_{\mu_1,G} + F(\nu_{\mu_2,G} - \nu_{\mu_3,G}).$$

  The values $\mu_1, \mu_2, \mu_3 \in \{1, 2, \ldots, N_p\}$ denote random non-repeated indexes of population vectors that are different from the target vector, so a population of a least 4 is necessary. The amplification of the differential variation $(\nu_{\mu_2,G} - \nu_{\mu_3,G})$ is controlled by real factor $0 < F < 2$.

- Crossover: To increase the diversity, a trial vector is $u_{i,G+1}$ is formed

$$u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ldots, u_{Di,G+1}). \tag{2.14}$$

  Each of the elements in equation (2.14) is defined by

$$u_{ji,G+1} = \begin{cases} \upsilon_{ji,G+1} \text{ if } randb(j) \leq CR \text{ or } j = rbr(i) \\ \nu_{ji,G} \text{ if } randb(j) > CR \text{ or } j \neq rbr(i) \end{cases}, \quad j = 1, 2, \ldots, D.$$

  We denote $randb(j)$ as the jth evaluation of a uniform random number generator with output values in $[0, 1]$. The crossover probability is defined by $CR \in [0, 1]$, determined by the user. To ensure that $u_{i,G+1}$ gets at least one parameter from $\upsilon_{i,G+1}$, an index $rnbr(i) \in \{1, 2, \ldots, D\}$ is chosen randomly.

- Selection: The trial vector $u_{i,G+1}$ is compared to the target vector $\nu_{i,G}$. If vector $u_{i,G+1}$ yields a smaller cost function value than $\nu_{i,G}$, then $\nu_{i,G+1} = u_{i,G+1}$; otherwise $\nu_{i,G+1} = \nu_{i,G}$.

2.6 Chapter Conclusions

In this chapter we explained all the concepts that are necessary to define and understand the proposed algorithms that we show in Chapter 3. It is important to remember that this chapter only featured concepts that were used in the development of the work presented in this thesis, leaving out all of those that were not used in subsequent chapters.

Chapter 3

Lift k-NN Forecasting

This chapter proposes modifications to some of the algorithms presented in Chapter 2, as well as explaining the overall algorithm that serves as the main work of this thesis. The main algorithm will be discussed further in this chapter. However, in order to make it easier for the reader to understand, we describe it from a general standpoint here. The time series is decomposed using lifting coefficients provided by the DE algorithm. We only take the approximation coefficients to produce a forecast using k-NN. We get the forecast error between the prediction and the real value, feeding it to DE so that the evolution process continues.

We begin the explanations with some changes done to k-NN in order to improve performance and efficiency, as well as describing the process used to find the parameters for constructing the database and defining the number of neighbors. Next, we describe the implementation of lifting, since there is not known library that has it. Finally, we explain the overall algorithm, piecing together all of the aforementioned tools.

3.1 Modifying k-NN

Just as described in Section 2.2, calculating the Euclidean distance for each individual in the database once a query arrives is a very time consuming task; even more so if we take into account that the work in this thesis involves evolving a function that uses NN, making a long but acceptable task into one that is unfeasible. One way to solve this

problem without incorporating new parameters, which would imply that more tunning is required and could result in more time spent, is to reduce the total number of Euclidean distances calculated once a query arrives, which could reduce the time needed linearly, and, while not making the process extremely efficient, it makes it feasible for the purposes of this thesis.

If the individuals database is large enough, one could think that is a good idea to discard older data and only consider the most recent half or quarter of it. This would in fact reduce the overall time and, in general, the more recent the data, the better. However, we will take advantage of our knowledge of where the time series comes from. We forecast short-term load time series provided by CENACE, which, by analyzing the data and using common knowledge that energy consumption tends to be similar at a given hour throughout the days, we know that is has a daily seasonality. This means that, for example, when predicting a value at $04:00$, seems almost pointless to look for individuals around $16:00$, and we should only look for individuals around the time of the day the query is made.

To make things clearer, we will make definitions in terms of the sample indexes, instead of using timestamps, just bearing in mind that, since every sample is 15 minutes ahead or behind the adjacent ones, samples corresponding to a same time of the day in different days will be separated by a multiple of 96 number of samples. This means that, given a sample at $r < 96$, we can define any sample z that was taken at the same time of the day as r by

$$z \equiv r \mod 96. \tag{3.1}$$

Assuming that we need to predict the next sample after the last one found in the database, being N the number of individuals or length of the database, we will define r as

$$r = 96 - (N \mod 96). \tag{3.2}$$

With the definition of equation (3.2), we can define a window frame w that we use to search around r, that is, we only calculate the Euclidean distance between the query and any individual at index z that holds

$$((z + w + r) \mod 96) \leq 2w. \tag{3.3}$$

Although it may seem confusing at a first glance, we are expressing the condition in terms of $2w$ in equation (3.3) because we want to consider w samples prior and after those defined in equation (3.1). If we took w out of the left side of the equation and subtracted it from the right side, we would have an issue with the preceding samples, since their indexes would have values within the range of $[-w, -1]$ and, for example, $-1 \mod 96 = 95$, which is most certainly a bigger number than w. Otherwise there is no point in making this modification.

For the sake of completeness, Algorithm 1 shows how the k-NN modification is implemented. Most of the lines refer to simple variable assignations based on the equations presented, however there are a few lines that need to be discussed. Line 3 defines an array named $Individuals$ whose elements consist of the Euclidean distance between an individual that holds the condition defined in equation (3.3) and the prediction associated with that individual, doing the process of filling the array in line 7. In line 10 we sort the $Individuals$ array by the distance value. Finally, we return the average of the predictions of the k considered individuals with the smallest distance.

3.1.1 NN parameter selection

Since we are using a very simple implementation of k-NN, as defined in Section 2.2, we only need to pay attention to two integer parameters: length of the individuals M and number of neighbors k. If we have a vague idea of where some integer parameters lie, then we can perform what is know as a grid search.

A grid search consists in trying out all the possible combinations of function parameters in a predefined search subspace and choosing the one that, in our case, yields the lowest output value. For example, if we were to find the optimal parameters for a function $f(x, y)$ searching in a subspace so that $x \in [I, I+1, \ldots, I+P]$ and $y \in [J, J+1, \ldots, J+L]$, then we would evaluate

$$f(I, J), f(I+1, J), \ldots, f(I+P, J), f(I, J+1), f(I+1, J+1), \ldots, f(I+P, J+L).$$

Algorithm 2 shows the implementation of the grid search algorithm to tune the parameters of k-NN. Line 2 defines the variable *best* that will be holding the lowest output

Algorithm 1: Change to how the k-NN are picked.

Input: query *query*, individuals database *database*, window frame w, number of neighbors k

Output: values *indexes* for the individuals length and number of neighbors that yield the best results within the grid

function FINDNEIGHBORS(query,database,w,k)
    $N \leftarrow length(database)$
    $Individuals \leftarrow NewArray()$
    $r \leftarrow 96 - (N \mod 96)$
    for $z \leftarrow 0; z < N$ **do**
        if $((z + w + r) \mod 96) \leq 2w$ **then**
            $Individuals.append([Euclidean(query, database[z][: -1]), database[z][-i]])$
        end if
    end for
    $Individuals.sortByDistance()$
    return $Mean(Individuals[:, 1][: k])$
end function

value of our target function; since we want to find the lowest value in the grid or subspace, we set the variable to have an initial value of infinite, being that any numeric value will be lower. Line 3 defines *indexes*, responsible for having the parameter values that yield the best results. Given that it is a fact that we will find a better output value than the initial, we set *indexes* to $[0, 0]$, using those parameter values as mere placeholders. Line 4 does not consider the last two individuals, since the last serves as a testing value and we make one prediction two steps ahead, that is, defining lead time $\tau = 2$. Line 7 uses a function to create the individuals database, based on equations (2.5) and (2.6). Line 8 uses a function that performs k-NN regression using the modification showed in Algorithm 1. Line 10 we use $ErrorMeasure$ as a placeholder for an error metric defined by the needs of the user. This means that *candidate* will have the forecast error between the real value and the value of our k-NN regression function, for each of the possible combinations. Lines 11 to 13 compare the output value and update it and the parameter values that yield it if necessary. Finally, we return the parameter values that achieved the best results.

With the resulting M we define the length of the individuals in our database and the length of the queries as well. The resulting k will be used when applying the modified k-NN rule, showed in Algorithm 1.

3.2 Redundant Lifting Implementation

Even though, by looking at Section 2.4, it is easy to implement the lifting scheme, there is no known library that does it, so, in order to make the work in this thesis fully reproducible by the reader, we will show a simple lifting implementation.

3.2.1 Lifting steps

We begin by defining how to apply the lifting steps to a signal, particularly the *Update* and *Predict* steps, since the *Split* step is done implicitly in the overall lifting algorithm. Algorithm 3 shows the *Predict* implementation. It consists in, given an input signal, defining the output as a signal constructed by adding to each element of the input its next one. Since there is no value after the last one, we get the last value of the output

Algorithm 2: Grid search for k-NN parameters.

Input: lower search bound for individuals length $initM$, upper search bound for individuals length $endM$, lower search bound for number of neighbors $initk$, upper search bound for number of neighbors $initk$, time series $times$, window frame w

Output: values $indexes$ for the individuals length and number of neighbors that yield the best results within the grid

```
1: function GRIDSEARCH(initM,endM,initk,endk,times,w)
2:     best ← ∞
3:     indexes ← [0, 0]
4:     historicTS ← times[: −2]
5:     for i ← initM; i < endM do
6:         for j ← initk; j < endk do
7:             indDB ← createDB(historicTS, i)
8:             query ← historicTS[−i :]
9:             forecast ← findNeighbors(query, indDB, w, j)
10:            candidate ← ErrorMeasure(forecast, times[−1])
11:            if candidate < best then
12:                best ← candidate
13:                indexes ← [i, j]
14:            end if
15:        end for
16:    end for
17:    return indexes
18: end function
```

signal as the original value from the input. Line 2 does this assignation.

Algorithm 3: Predict step for lifting.

Input: signal *signal*

Output: sum of input signal with itself shifted one to the right *prediction*

```
1: function PREDICT(signal)
2:     prediction ← signal
3:     prediction[: −1] ← signal[: −1] + signal[1 :]
4:     return prediction
5: end function
```

Algorithm 4 shows the *Update* step, being analogous to the *Predict* step, however we add to each sample of the signal its previous value, instead of its next.

Algorithm 4: Update step for lifting.

Input: signal *signal*

Output: sum of input signal with itself shifted one to the left *renew*

```
1: function UPDATE(signal)
2:     renew ← signal
3:     renew[1 :] ← signal[1 :] + signal[: −1]
4:     return renew
5: end function
```

3.2.2 Lifting decomposition and reconstruction

With the lifting steps covered, we now explain how to implement redundant lifting. As explained in Subsection 2.4.1, redundant lifting consists in regular lifting with just an extra lifting decomposition with a shift in the input signal. Since we need to test the lifting scheme at various capacities, that is, with different number of *Predict*/*Update* pairs, we designed the algorithm to be able to perform the necessary number of steps given the length of the lifting coefficients. Since the repeatable steps always come in a pair, we will always

have an odd number of lifting coefficients greater than 3, due to the last lifting coefficient being the scaling factor.

Algorithm 5 shows how to implement lifting decomposition to a signal. The algorithm is a direct translation from the definitions in Chapter 2.4, but it is important to explain lines 3 and 4, since there is room for confusion. We stated that the details d would be defined as the prediction error for the odd signal coefficients, being the approximations s similar but in terms of the even ones; however line 3 defines d as the values of the even indexes of the signal, proceeding similarly for s using the odd indexes in line 4. This is due to the fact that, while in most programming languages we start numbering the indexes stating from 0, from a positional standpoint and based on the mathematical definition, the first value, found at position 1, is the one corresponding to index 0, making this sort of shift for the rest of the index values.

Algorithm 5: Lifting decomposition to a signal.

Input: signal $signal$, lifting coefficients $coeffarray$

Output: approximation coefficients s, detail coefficients d

1: **function** LIFT(signal, coeffarray)
2: $coefflen \leftarrow length(coeffarray)/2$
3: $d \leftarrow signal[0 :: 2]$
4: $s \leftarrow signal[1 :: 2]$
5: **for** $i \leftarrow 0; i < coefflen$ **do**
6: $d \leftarrow d - coeffarray[2 * i] * predict(s)$
7: $s \leftarrow s + coeffarray[2 * i + 1] * update(d)$
8: **end for**
9: $s \leftarrow s * coeffarray[: -1]$
10: $d \leftarrow d/coeffarray[: -1]$
11: **return** s, d
12: **end function**

Even though in Subsection 2.4.1 we say that, in order to apply redundant lifting, we need to use the $lift$ function but feed it with the signal being shift forward one sample,

we define a new function called $liftShift$ that receives the same inputs as $lift$, but d is made out of the values at the odd indexes and s out of the even ones. We do this because, although mathematically correct, shifting the signal one sample would imply not having an even amount of total samples, making us unable to perform the LWT.

With both of the $lift$ and $liftShift$ functions, we can implement the redundant lifting. Algorithm 6 shows the implementation of redundant lifting, which just directly follows the aforementioned definition.

Algorithm 6: Redundant lifting decomposition to a signal.

Input: signal $signal$, lifting coefficients $coeffarray$

Output: approximation coefficients $apro$, detail coefficients $deta$

1: **function** REDLIFT(signal, coeffarray)
2: $aproe, detae \leftarrow lift(signal, coeffarray)$
3: $aproo, detao \leftarrow liftShift(signal, coeffarray)$
4: $apro \leftarrow NewArray(Length(signal))$
5: $deta \leftarrow NewArray(Length(signal))$
6: $apro[0 :: 2] \leftarrow aproo$
7: $apro[1 :: 2] \leftarrow aproe$
8: $deta[0 :: 2] \leftarrow detao$
9: $deta[1 :: 2] \leftarrow detae$
10: **return** $apro, deta$
11: **end function**

Being done with the decomposition, we now move to the inverse wavelet transform through lifting. Just as defined in Section 2.4, we take as inputs the approximation, detail and lifting coefficients, and output the reconstructed signal. Algorithm 7 shows the reconstruction process for a signal, just as described.

As we described in Subsection 2.4.1, to apply an inverse redundant lifting process, we just need to split the approximation and detail coefficients and apply the regular inverse lifting process to just the even samples, corresponding to the odd array indexes. In Algorithm 8 we show said process.

Algorithm 7: Lifting reconstruction of a signal.

Input: approximation coefficients *approx*, detail coefficients *detail*, lifting coefficients $coeffarray$

Output: reconstructed signal *reconst*

1: **function** INVLIFT(approx, detail, coeffarray)
2: $s \leftarrow approx/coeffarray[:-1]$
3: $d \leftarrow detail * coeffarray[:-1]$
4: $coefflen \leftarrow length(coeffarray)/2$
5: **for** $i \leftarrow coefflen; i \geq 0; i--$ **do**
6: $s \leftarrow s - coeffarray[2*i+1] * update(d)$
7: $d \leftarrow d + coeffarray[2*i] * predict(s)$
8: **end for**
9: $reconst \leftarrow NewArray(Length(d)*2)$
10: $reconst[0::2] \leftarrow d$
11: $reconst[1::2] \leftarrow s$
12: **return** *reconst*
13: **end function**

Algorithm 8: Redundant lifting reconstruction of a signal.

Input: approximation coefficients *approx*, detail coefficients *detail*, lifting coefficients $coeffarray$

Output: reconstructed signal *reconst*

1: **function** INVREDLIFT(approx, detail, coeffarray)
2: $aproe \leftarrow approx[1::2]$
3: $detae \leftarrow detail[1::2]$
4: $reconst \leftarrow invLift(aproe, detae, coeffarray)$
5: **return** *reconst*
6: **end function**

Having described the implementation for the decomposition and reconstruction schemes using lifting, we are just left to explain an implementation that allows us to decompose and reconstruct for a given level l. In order to implement the multilevel decomposition, we just need to remember that the scheme iterates over the approximation coefficients of the previous level of the one resulting from our decomposition. Although simple, for the sake of completeness, we show in Algorithm 9 that implementation. We define in line 2 the $decompcoeff$ array that stores the last level approximation coefficients and the detail coefficients for each level. Using s_i and ρ_i to denote the approximation and detail coefficients for decomposition level i, and defining the target decomposition level l, the function will return an array in the form

$$(s_l, \rho_l, \rho_{l-1}, \ldots, \rho_1).$$

We output the decomposition coefficients in this manner to follow the standard used by all the libraries that implement the wavelet transform, whether it is in its continuous or discrete variant. For the same reason, this is how the input for the inverse must be given.

Algorithm 9: Multilevel redundant lifting decomposition to a signal.

Input: signal $signal$, lifting coefficients $coeffarray$, decomposition level $level$

Output: array of details and approximation coefficients $decompcoeff$

```
1:  function MULTIREDLIFT(signal, coeffarray, level)
2:      decompcoeff ← NewArray()
3:      targetsignal ← signal
4:      for i ← 0; i < level do
5:          targetsignal, detail ← redLift(targetsignal, coeffarray)
6:          decompcoeff.insertAtBeginning(detail)
7:      end for
8:      decompcoeff.insertAtBeginning(targetsignal)
9:      return decompcoeff
10: end function
```

To apply the multilevel reconstruction using the lifting scheme, we perform a re-

construction using the approximation and detail coefficients of the last level, and iterate using the output as the approximation coefficients and the corresponding detail coefficients of the next level to reconstruct. Algorithm 10 shows the implementation of the aforementioned process. It is worth nothing that the decomposition level does not need to be specified, since it is equal to the amount of detail coefficients sets, corresponding to the overall number of elements in the decomposition coefficients array minus one.

Algorithm 10: Multilevel redundant lifting reconstruction of a signal.

Input: array of details and approximation coefficients $decompcoeff$, lifting coefficients $coeffarray$

Output: reconstructed signal $approx$

1: **function** MULTINVREDLIFT(decompcoeff, coeffarray)
2: $approx \leftarrow decompcoeff[0]$
3: $decomplen \leftarrow Length(decompcoeff)$
4: **for** $i \leftarrow 1; i < decomplen$ **do**
5: $approx \leftarrow invRedLift(approx, decompcoeff[i], coeffarray)$
6: **end for**
7: **return** $approx$
8: **end function**

3.3 Forecasting Algorithm

Now that we have covered the definitions and the implementations of all the necessary tools for the work presented in this thesis, except the DE implementation, we can explain the overall and main algorithm. For DE, we use the *SciPy* library [Virtanen et al., 2020]. In order to make the explanation easier, we show in Figure 3.1 a block diagram which depicts the process that we describe next.

Even though experiments were made using the scheme, commenting more on the topic in Chapter 4 and 5, the main work in this thesis does not forecast using all the decomposition coefficients and then performs the inverse transform; instead we use the wavelet

transform as a denoiser. This means that we only work with the last level approximation coefficients, make the forecasts and output them as general forecasts of the input time series. Normally using a wavelet transform as a denoiser can be problematic if the frequency components that we are leaving out are significant to the overall behavior of the time series; however we use DE to tune the lifting coefficients so that the denoising just leaves out what can be considered, in fact, as noise. Given that DE evolves real valued parameters, such as the lifting coefficients, any wavelet can be expressed through lifting, lifting also makes non-linear wavelet transforms, and lifting is not a unique process, using both DE and lifting in tandem seems only fitting.

First, we scale the original input time series so that its values are in the range $[0, 1]$. We do this to prevent overflow and other problems associated with great numeric values, since electricity load time series have values in the tens of thousands. Then, we decompose the scaled time series with lifting, with the lifting coefficients being provided by DE. We take just the last level approximation coefficients and create the individuals database with them. We use k-NN regression to get the forecasts of the approximation coefficients. Since lifting and some wavelet transforms do not preserve energy, we scale the approximation coefficients A to match the values of the $[0, 1]$ scaled time series y_s. We do this by multiplying the approximation coefficients by a factor v defined as

$$v = \frac{\max(y_s)}{\max(A)}. \tag{3.4}$$

One of the advantages of scaling in $[0, 1]$ is that, since there are only non-negative values, we do not need to consider an offset in equation (3.4). Next, we apply the inverse scaler to change the energy preserved forecasts values from $[0, 1]$ to those corresponding the input time series and output them as overall forecasts of the time series. Finally, we obtain the fitness value for DE by calculating an error measure between the forecasts and the real expected values, feeding that value to DE. The process ends when the maximum number of iterations is reached or when no new set of coefficients are generated, known as convergence. When the process ends, we are left out with the best forecast made and with the lifting coefficients that ultimately produced it.

Even though the process is very simple and straightforward, in order to make

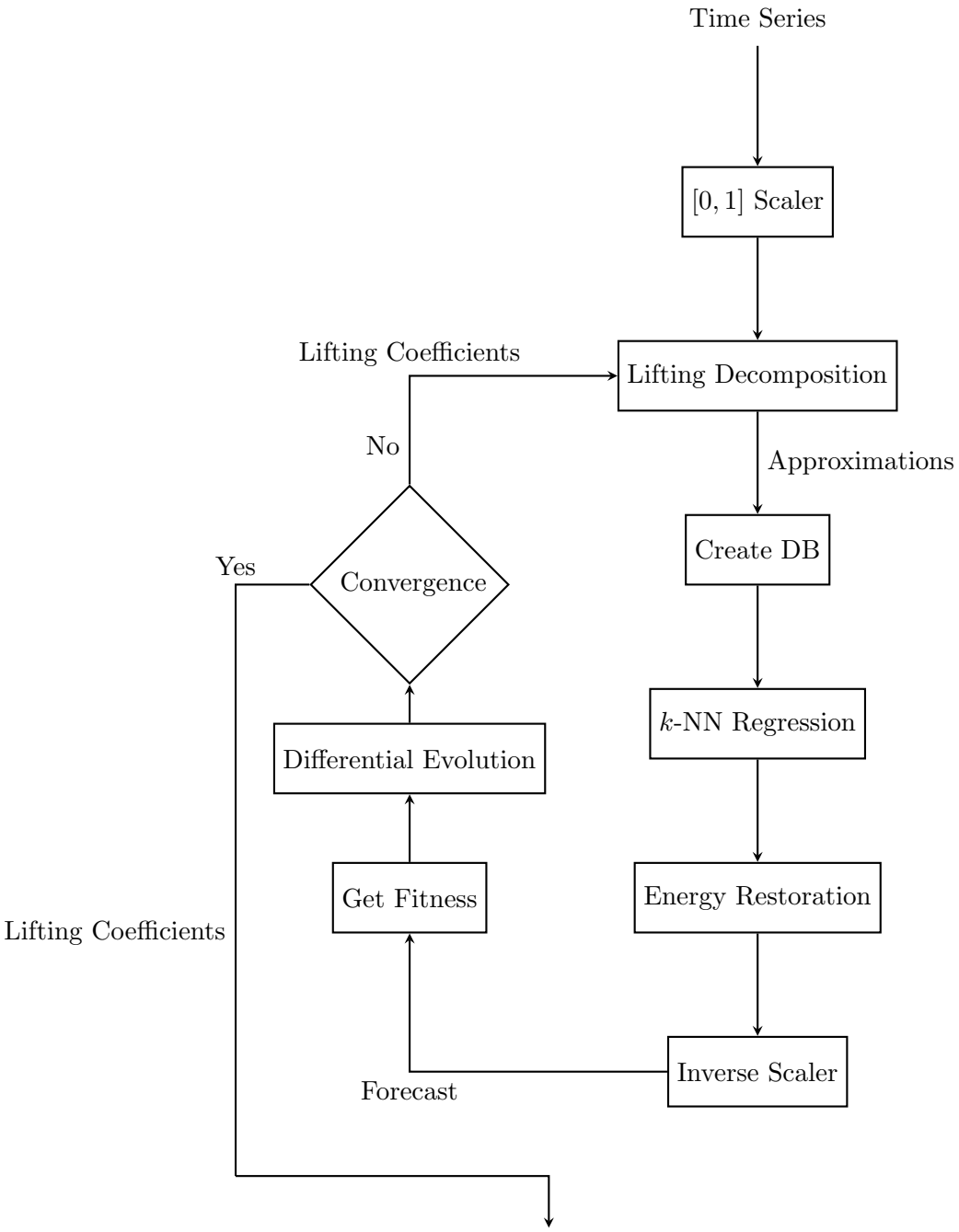

Figure 3.1: Overall evolving process.

this work reproducible and for the sake of completeness, we show in Algorithm 11 the implementation of the main function that returns the fitness value that DE uses. We are not including DE, again, because we use a library for it. It receives as input the time series, lifting coefficients, decomposition level, length of individuals, and number of neighbors. It is worth noting that Algorithm 11 just makes one prediction, but we have faith in the reader in order to slightly modify it and be able to produce an arbitrary number of forecasts, which is precisely the case showed in Chapter 4.

Algorithm 11: Forecasting with k-NN and lifting.

Input: time series $times$, lifting coefficients $coeffarray$, decomposition level $level$, query and individuals length m, number of neighbors k

Output: error $fitness$ between the forecast and the real value

1: **function** LIFTNN(times, coeffarray, level, m, k)
2: $scaledts \leftarrow Scaler(times[:-2])$
3: $maxts \leftarrow Max(scaledts)$
4: $appro \leftarrow multiRedLift(scaledts, coeffarray, level)$
5: $maxappro \leftarrow Max(appro)$
6: $indDB \leftarrow createDB(appro, m)$
7: $query \leftarrow appro[-m:]$
8: $forecast \leftarrow NNRegression(query, indDB, k)$
9: $forecast \leftarrow forecast * maxts/maxappro$
10: $forecast \leftarrow InverseScaler(forecast)$
11: $fitness \leftarrow ErrorMeasure(forecast, times[-1])$
12: **return** $fitness$
13: **end function**

3.4 Chapter Conclusions

In this chapter we presented algorithms that allow us to perform a more efficient k-NN regression scheme. We also described how we can find the optimal k-NN hyper-

parameters within a fixed region. We presented a lifting implementation that successfully decomposes a signal without decimation. We gave an algorithm that combines k-NN and lifting to forecast. We also explained how the evolution of the lifting coefficients is performed. We show in Chapter 4 the results of using the algorithms presented in this chapter in short-term load time series.

Chapter 4

Results

The forecasting task used in the experiments consists in seven time series of 35421 samples taken every 15 minutes, from which 35421 are used as history and the last 92 are used as a test set. After forecasts for each of the seven time series are obtained, they are added up together and make the final and main forecast. This means that we end up with seven partial forecasts and a main one. The decision to use this particular task was made due to its complexity, since it would serve as a precedent to tackle simpler forecasting schemes.

Before we begin explaining the specific tests and their respective results, it is important to address the characteristics of the computer in which the testing was made, since one of the measures this thesis reports is the amount of time spent in the various processes. For the experiments, an MSI GE62 6QF Apache Pro laptop was used, with an Intel Core i7 6700HQ processor at $3.50\,GHz$ with $6\,MB$ in cache and $12\,GB$ of DDR4 RAM at $2133\,MHz$.

This chapter discusses the search experiment used to find the k-NN parameters that produced the best results. Next, we address the evolution experiments conducted in order to find the lifting coefficients that improved the k-NN forecasting. Then, we show the forecasting tasks done with the evolved lifting coefficients. We also show some results using k-NN alone and with the aid of the regular DWT, as we need them to make a direct comparison with the ones obtained. Finally, we mention some failed experiments that did

not produce the expected results, in case the reader decides to delve in this topic, as some considerations may be taken in mind.

Just as mentioned when explaining Algorithm 11, we need to define an error measurement used by the algorithms responsible for finding parameters and to be able to compare the various forecasting schemes. Since it is the most common measurement used in forecasting research, we will use the *Mean Absolute Percentage Error* (MAPE), which, using the notation from 2.1, is defined as:

$$E = \frac{1}{T}\sum_{t=1}^{T}\left|\frac{y_t - \hat{y}_t}{y_t}\right|.$$

Normally, it is possible to have a division by zero in the case of $y_t = 0$. However, unless extraordinary circumstances happen, that is not the case with short-term load time series.

From previous experiments made by various members of the faculty using these same time series, involving ANNs and Nearest Neighbors with DE evolved hyper-parameters, the best MAPE measurement and target value to beat with the proposed schemes was 0.6475397. Making the goal since the definition of the parameters for k-NN regression to be as close as possible to that value or to improve it.

4.1 Grid Search and Evolution Experiments

Before we can make k-NN forecasting with the aid of evolved lifting coefficients, we first need to find parameters for both of them that produce results good enough to outperform the aforementioned cases that do not use this particular tandem. We begin by describing the process for the k-NN parameters, because it is not a good idea to have a mediocre k-NN forecasting, leaving all the hard work to lifting. Even though it is documented that the wavelet transform can help, it does not make drastic increments in performance.

4.1.1 Grid search results

Before beginning the grid search to find the length M of the vectors of the individuals database and queries, and the number of considered neighbors k, we first fixed $k = 1$, since in [Cover and Hart, 1967] it is proven that such consideration is capable of

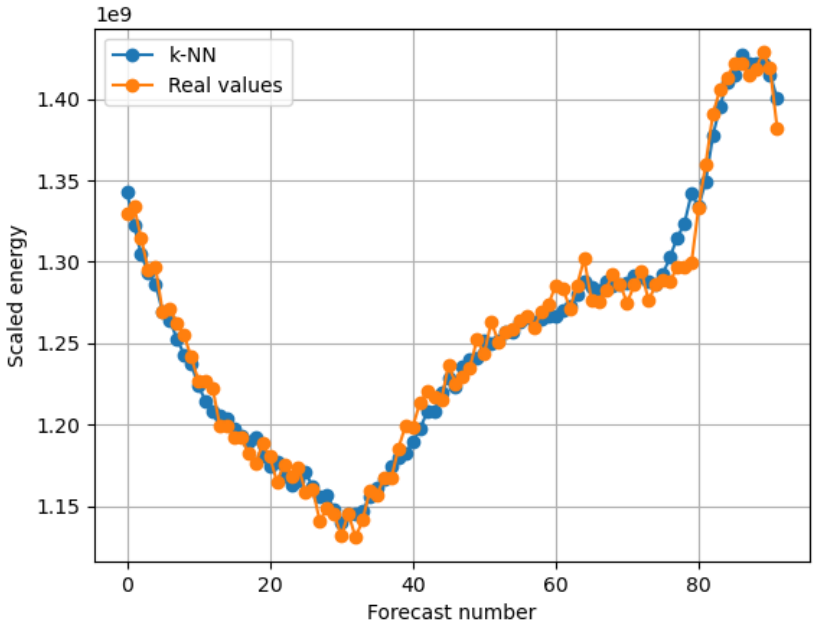

Figure 4.1: Main forecasts with tuned k-NN.

yielding acceptable results, and tested values from $M = 5$ to a number that gave worse results than the previous one tested, incrementing M by 5. This means that, considering a value $M = m_f + 5$ that performed worse than m_f, the succession of tested values for M can be seen as $M = 5, 10, \ldots, m_f, m_f + 5$. We did this in order to reduce the search space since, due to the curse of dimensionality, this type of searches can take significant time. Using the previous notation, we found $m_f = 60$, so the decision to search from 50 to 70 was made. For k, the search ranged from $k = 1$ to $k = 10$. This means that a total of $21 \cdot 10 = 210$ combinations were considered. The winning pair was $M = 61$ and $k = 5$, with a MAPE value of 0.6374744. In Figure 4.1 we can see the main forecasts, that is the sum of the seven components, that yielded the aforementioned result. It is important to note that, given that we are using sensible data, the values of the time series have been scaled.

4.1.2 DE results

We used DE to obtain the best possible the lifting coefficients. Since the number of individuals examined depends on the dimension of the parameter vector, we started with just 3 lifting coefficients. Based on [Grasemann and Miikkulainen, 2004] and [Grasemann and Miikkulainen, 2005], we decided to define the boundaries for the update and predict coefficients to $[-1, 1]$ and the one for the scaling factor to $[-2, 2]$. The default parameters for DE given by the SciPy library were used, just changing the maximum number of iterations to 55 and the population size to 8. Since it does not take significant time and it is easier to perform the denoising throughout several decomposition levels, a level 3 decomposition was implemented. After 17 iterations, which corresponds to 408 analyzed individuals, the values converged and, therefore, the evolution process halted. The desired value was not obtained, in fact, resulted worse than without using lifting. A second evolution experiment was conducted, with different initial individuals, having a similar result.

Since increasing the number of lifting steps, increases its representation capacity, the decision to evolve 5 lifting coefficients was made. The same DE parameters from the previous experiment were used and, after 49 iterations, which corresponds to 1960 analyzed individuals, the process halted on a convergence. However, the goal was met with a MAPE value of 0.6337626. In Figure 4.2 we show the main forecasts using k-NN aided with lifting, using the evolved coefficients.

Since with the aid of the evolved lifting coefficients all the other results were outperformed, there is interest in seeing how the decomposition of the signals was made. Given that the main time series was not decomposed, one of the other seven time series was used to show the decomposition. In Figures 4.3, 4.4, 4.5 and 4.6, we show the first, second and third level detail coefficients, and the third level approximation coefficients, respectively. Additionally, we show in Figure 4.7 the original time series that was decomposed.

4.2 Results Comparison

Even though we already stated that the scheme that uses evolved lifting coefficients was the one that performed the best, we will present the MAPE values for all of the partial

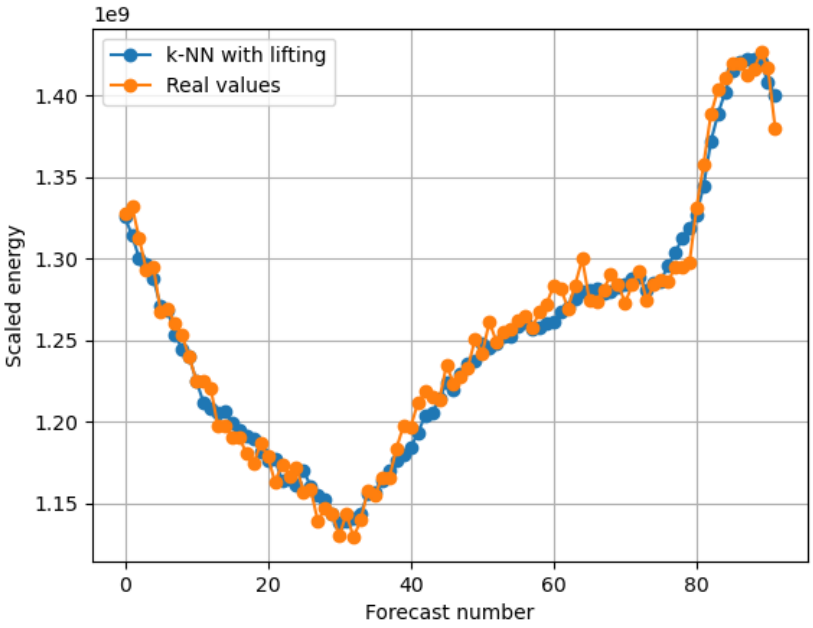

Figure 4.2: Main forecasts using lifting and tuned k-NN.

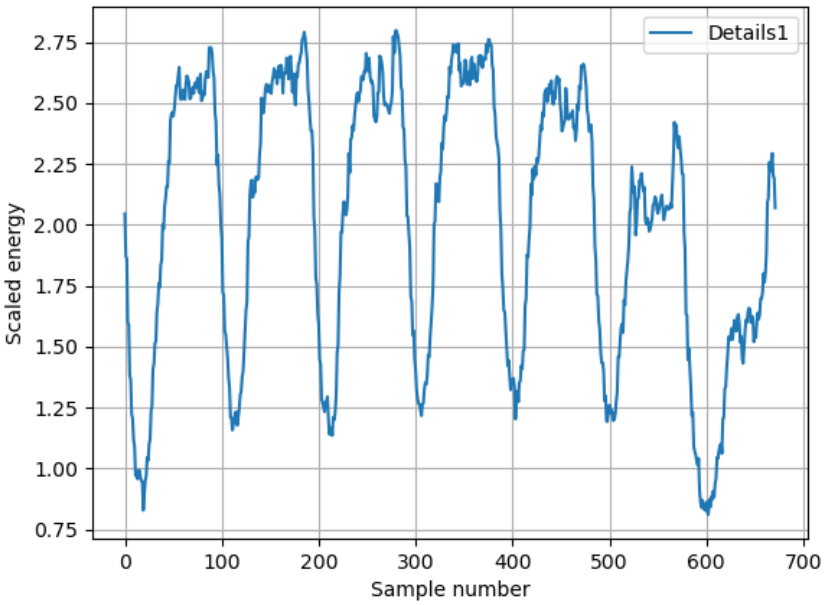

Figure 4.3: First level detail coefficients.

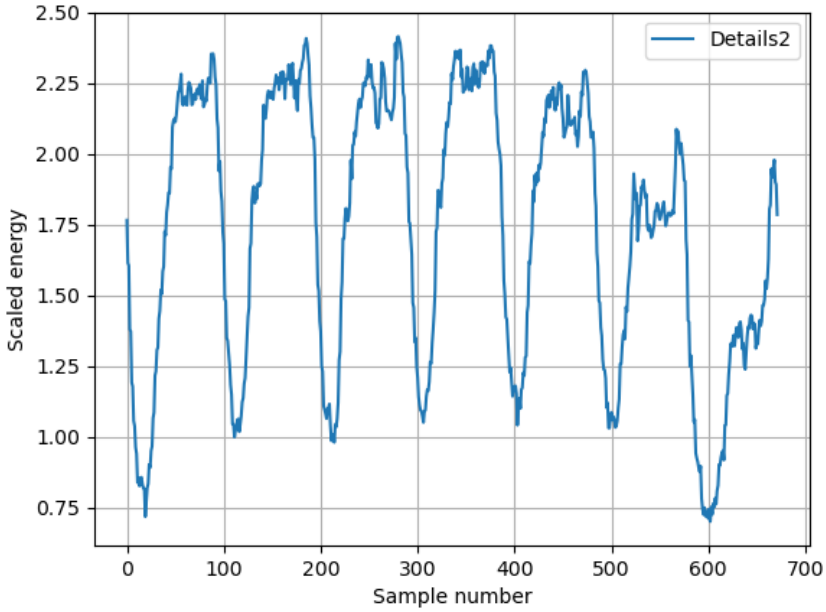

Figure 4.4: Second level detail coefficients.

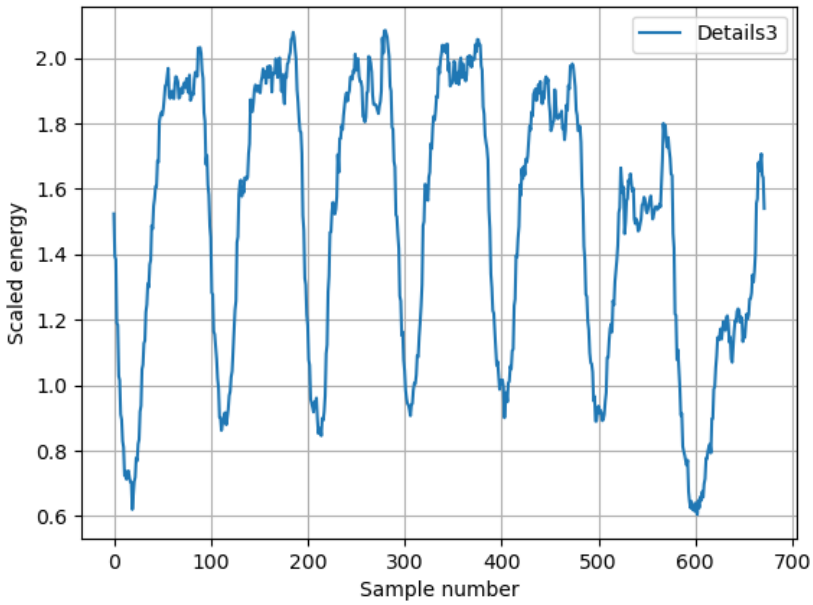

Figure 4.5: Third level detail coefficients.

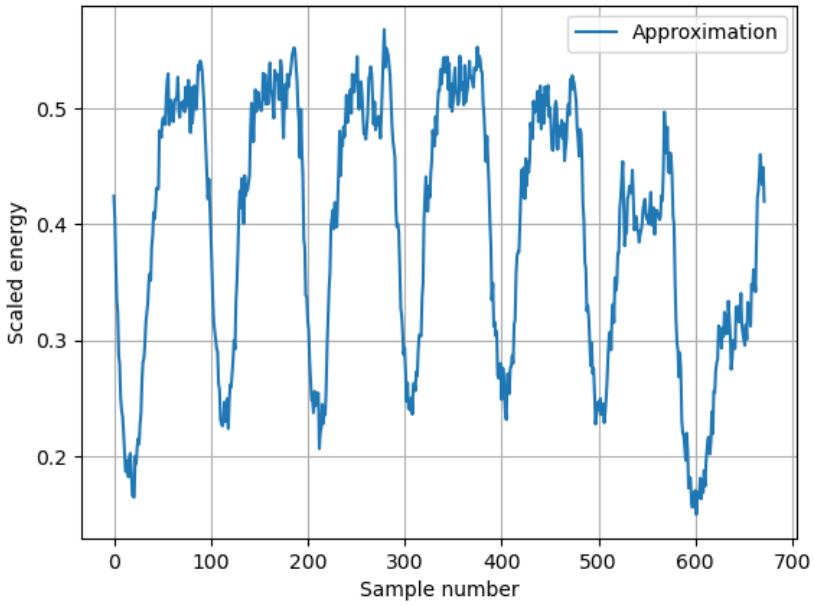

Figure 4.6: Third level approximation coefficients.

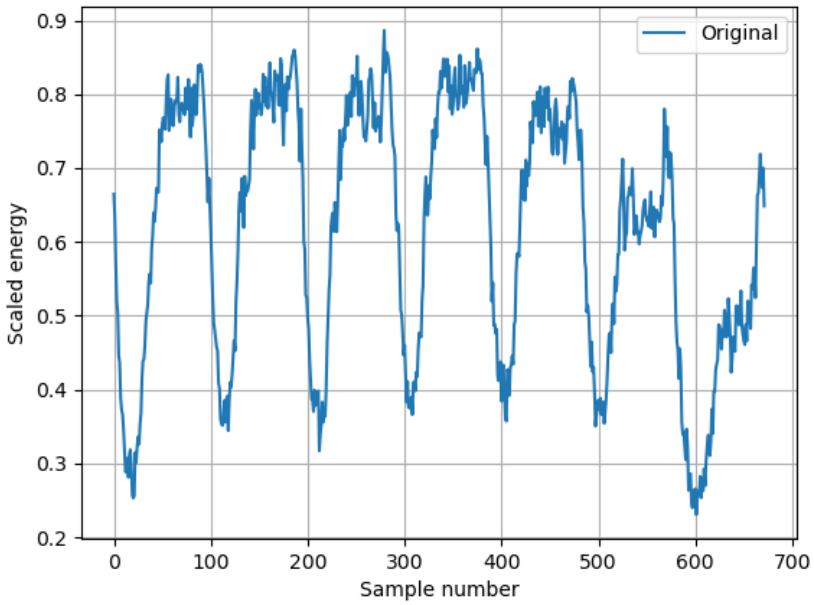

Figure 4.7: Original time series.

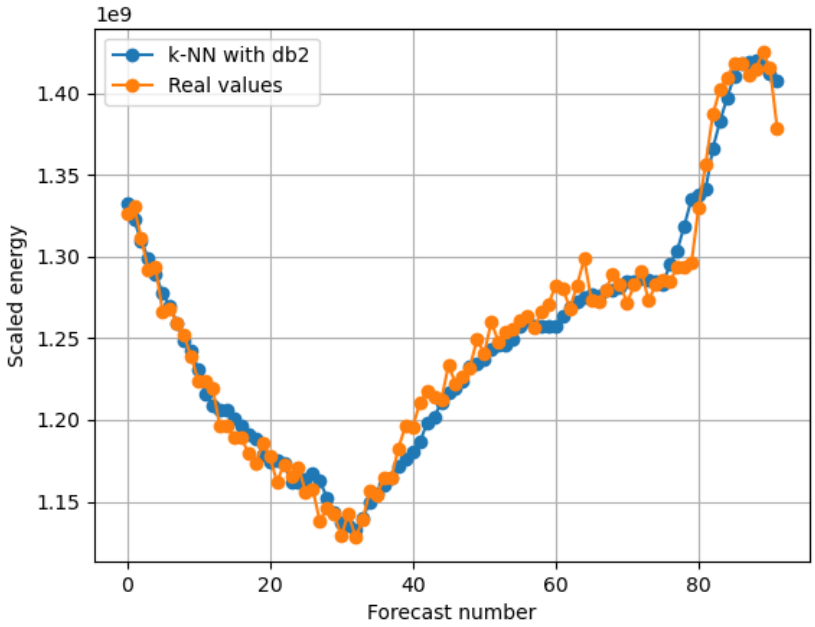

Figure 4.8: Main forecasts using db2 and tuned k-NN.

and main forecasts, as well as the time each of the schemes needed.

In order to justify the proposal of this thesis with results, we used traditional wavelets to see if they could outperform the lifting results. We tried the Daubechies wavelet family as well as the FBI wavelet. To do this, we used the PyWavelets library [Lee et al., 2019]. Of all the ones considered, the one that provided the best results was the order 2 Daubechies wavelet (db2), and performed the best when only using just one decomposition level. However, it was not able to even meet the error criteria defined by the advisor. It is important to note that this is also a denoising scheme, just as with lifting. In Figure 4.8 we show the main forecasts using that wavelet. Table 4.1 shows a comparison of the results obtained by using Daubechies wavelets from order 1 to 5. More Daubechies wavelets were tested, but their error increased as the order did.

Table 4.2 shows the performance of k-NN regression alone, k-NN regression using the db2 wavelet, and k-NN regression using lifting with evolved coefficients. The only time series were aided by lifting scheme has a greater MAPE value is in the fourth time series,

	Daubechies wavelets				
Time Series	Order 1 (db1)	Order 2 (db2)	Order 3 (db3)	Order 4 (db4)	Order 5 (db5)
1st	2.3227990	**1.9160500**	2.1322805	2.3481188	2.5712097
2nd	2.2085885	**1.7996213**	2.0864264	2.1306598	2.1306258
3rd	2.0944288	**2.0321481**	2.5594039	3.0444270	3.5019938
4th	2.7364407	1.4411677	**1.2768998**	1.3060665	1.5089668
5th	2.7606201	**1.7703327**	2.0083601	2.0965425	2.2691137
6th	1.8296414	**1.5238273**	1.5241821	1.8209926	2.2528962
7th	2.0423074	**1.6897004**	1.9228182	2.2751868	2.5963218
Main	1.4018464	**0.7121444**	1.0288592	1.2334200	1.4569735

Table 4.1: MAPE values using various Daubechies wavelets with k-NN.

	Schemes		
Time Series	k-NN	k-NN + db2	k-NN + lifting
1st	1.8636617	1.9160500	**1.8255230**
2nd	1.8258686	1.7996213	**1.7646766**
3rd	1.8999023	2.0321481	**1.8286672**
4th	2.7630127	**1.4411677**	1.5363824
5th	1.7102076	1.7703327	**1.6357255**
6th	1.5633339	1.5238273	**1.5222299**
7th	1.8226193	1.6897004	**1.5631419**
Main	0.6374744	0.7121444	**0.6337626**

Table 4.2: MAPE values of the considered schemes.

although not by much.

In terms of the time needed to make all the forecasts, as expected, just using k-NN alone had the best time with 30.1 s; it took more time when decomposing the time series, taking 98.0 s for the scheme with the db2 wavelet, and 97.2 s for the one that uses evolved lifting.

4.3 Unsuccessful Attempts

Even though there were experiments that did not improve the MAPE value, they can give some insight in case this topic is further researched. Aside from the 3 coefficients experiment, the original idea was to make forecasts on each of the 4 components and then use

the inverse transform to get the final forecasts for each time series, since [Conejo et al., 2005] proposes this approach. This, however, did not produce acceptable results when doing the evolution process. Several attempts were made, going from 3 coefficients to 9 coefficients, and increasing the population. The maximum number of iterations was irrelevant, since there was always the case of convergence, but unable to satisfy the error goal. Since a lot of time went in the process and 9 lifting coefficients has more than enough representation capacity, the decision to use the lifting decomposition as a denoiser was rather adopted.

4.4 Chapter Conclusions

In this chapter we explained the experiments conducted to get the results of this thesis. We also provided the resulting hyper-parameters for k-NN. We addressed the experiments conducted to find the lifting coefficients that yielded an error that met the defined criteria. We compared various results within the Daubechies family, and also compared results of different schemes. We gave a brief summary of some experiments that did not produce results that met the criteria in order to provide the reader with more information, in case they decide to delve in this topic. We will show in Chapter 5 the general conclusions of this thesis, along with propositions for future work.

Chapter 5

Conclusions

This chapter presents general conclusions that consist in an interpretation of the obtained results that are shown in Chapter 4. It also features some ideas that may serve to conduct future research, based on the work of this thesis.

5.1 General Conclusions

In this thesis, we were able to define a k-NN regression based forecasting method that was able to outperform the best results of previous research for a given set of time series, while reducing the amount of time needed to make said forecasts, once the lifting coefficients were found. We successfully evolved lifting coefficients that, when used in tandem with the k-NN regression method as a denoising scheme, were able to best the results that were obtained without lifting and even those with the use of traditional wavelets. Adding to this, the winning decomposition does not show the behavior one would expected form a first generation wavelet, most likely being a non-linear wavelet, only being able to be implemented through lifting.

This work is also the first attempt, to the date this thesis was written, to evolve wavelets through lifting for forecasting tasks, and is one of the few to evolve lifting coefficients that achieve desirable results in general. Additionally, we were able to corroborate the lifting properties. In particular, since there was no optimization in the coding of the lifting process and was able to make the decomposition faster than highly optimized libraries, we

also showed that there are many advantages of using lifting instead of traditional DWT.

5.2 Future Work

We present some promising ideas that may serve as areas of potential research, and may lead to fruitful results:

1. Making forecasts on each component was not successful. This may be due to the fact that there is no fitting process in k-NN regression, so there is little apparent benefit on simplifying time series when used with it. However, forecasting algorithms that have a fitting process may benefit a lot from using this scheme.

2. Even if lifting is used as a signal denoiser, more forecasting schemes should be studied to see if there is a better payoff than with k-NN regression.

3. Although the error criteria was met, evolving more parameters as well as conducting experiments with more iterations and population may lead to even better results. It is a good idea to increase the wavelet representation capacity, as well as evolving the decomposition level. It is also worth considering properly evolving the k-NN parameters and introducing others that may improve the forecasts.

4. Time series of different domains have different behaviors, so it is an interesting idea to see if evolving wavelets through lifting can also help forecasting time series that have significantly different frequency components.

Bibliography

[Anandakumar et al., 2019] Anandakumar, Kumar, A., Kale, R., Babu, B., Sathishkumar, U., Reddy, G., and Kulkarni, P. (2019). A hybrid-wavelet artificial neural network model for monthly water table depth prediction. *Current science*, 117:1475–1481.

[Bunn and Farmer, 1985] Bunn, D. and Farmer, E. D. (1985). Comparative models for electrical load forecasting.

[Claypoole et al., 1998] Claypoole, R. L., Baraniuk, R. G., and Nowak, R. D. (1998). Adaptive wavelet transforms via lifting. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, volume 3, pages 1513–1516 vol.3.

[Coifman and Wickerhauser, 1992] Coifman, R. R. and Wickerhauser, M. V. (1992). Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory*, 38(2):713–718.

[Conejo et al., 2005] Conejo, A. J., Plazas, M. A., Espinola, R., and Molina, A. B. (2005). Day-ahead electricity price forecasting using the wavelet transform and arima models. *IEEE transactions on power systems*, 20(2):1035–1042.

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

[Daubechies, 1992] Daubechies, I. (1992). *Ten lectures on wavelets*, volume 61. Siam.

[Daubechies and Sweldens, 1998] Daubechies, I. and Sweldens, W. (1998). Factoring

wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications*, 4(3):247–269.

[Erba et al., 2001] Erba, M., Rossi, R., Liberali, V., and Tettamanzi, A. G. (2001). Digital filter design through simulated evolution. In *Proceedings of the 15th European Conference on Circuit Theory and Design, ECCTD*, volume 1, pages 137–140. Citeseer.

[Fan et al., 2019] Fan, G.-F., Guo, Y.-H., Zheng, J.-M., and Hong, W.-C. (2019). Application of the weighted k-nearest neighbor algorithm for short-term load forecasting. *Energies*, 12(5):916.

[Fix, 1951] Fix, E. (1951). *Discriminatory analysis: nonparametric discrimination, consistency properties*. USAF School of Aviation Medicine.

[Fix and Hodges Jr, 1952] Fix, E. and Hodges Jr, J. L. (1952). Discriminatory analysis-nonparametric discrimination: Small sample performance. Technical report, CALIFORNIA UNIV BERKELEY.

[Flores et al., 2019] Flores, J. J., González, J. R. C., Farias, R. L., and Calderon, F. (2019). Evolving nearest neighbor time series forecasters. *Soft Computing*, 23(3):1039–1048.

[Grasemann and Miikkulainen, 2004] Grasemann, U. and Miikkulainen, R. (2004). Evolving wavelets using a coevolutionary genetic algorithm and lifting. In Deb, K., editor, *Genetic and Evolutionary Computation – GECCO 2004*, pages 969–980, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Grasemann and Miikkulainen, 2005] Grasemann, U. and Miikkulainen, R. (2005). Effective image compression using evolved wavelets. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

[Hill et al., 2001] Hill, Y., O'Keefe, S., and Thiel, D. (2001). An investigation of wavelet design using genetic algorithms. In *Microelectronic Engeneering Research Conference*.

[Hippert et al., 2001] Hippert, H. S., Pedreira, C. E., and Souza, R. C. (2001). Neural networks for short-term load forecasting: a review and evaluation. *IEEE Transactions on Power Systems*, 16(1):44–55.

[Hopper et al., 1993] Hopper, T., Brislawn, C., and Bradley, J. (1993). Wsq gray-scale fingerprint image compression specification. *Federal Bureau of Investigation, Criminal Justice Information Services, Washington, DC, USA, Tech. Rep. IAFIS-IC-0110-V2*.

[Kantz and Schreiber, 2004] Kantz, H. and Schreiber, T. (2004). *Nonlinear time series analysis*, volume 7. Cambridge university press.

[Lankhorst and Laan, 1995] Lankhorst, M. and Laan, M. (1995). Wavelet-based signal approximation with genetic algorithms. volume 43, pages 237–255.

[Lee et al., 1999] Lee, A., Ahmadi, M., Jullien, G. A., Lashkari, R. S., and Miller, W. C. (1999). Design of 1-d fir filters with genetic algorithms. In *ISSPA '99. Proceedings of the Fifth International Symposium on Signal Processing and its Applications (IEEE Cat. No.99EX359)*, volume 2, pages 955–958 vol.2.

[Lee et al., 2019] Lee, G. R., Gommers, R., Waselewski, F., Wohlfahrt, K., and O'Leary, A. (2019). Pywavelets: A python package for wavelet analysis. *Journal of Open Source Software*, 4(36):1237.

[Liu and Wechsler, 2000] Liu, C. and Wechsler, H. (2000). Evolutionary pursuit and its application to face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):570–582.

[Monro and Sherlock, 1997] Monro, D. M. and Sherlock, B. G. (1997). Space-frequency balance in biorthogonal wavelets. In *Proceedings of International Conference on Image Processing*, volume 1, pages 624–627. IEEE.

[Montgomery et al., 2015] Montgomery, D. C., Jennings, C. L., and Kulahci, M. (2015). *Introduction to time series analysis and forecasting*. John Wiley & Sons.

[Osório et al., 2014] Osório, G., Matias, J., and Catalão, J. (2014). Hybrid evolutionary-adaptive approach to predict electricity prices and wind power in the short-term.

[Pai and Hong, 2005] Pai, P.-F. and Hong, W.-C. (2005). Support vector machines with simulated annealing algorithms in electricity load forecasting. *Energy Conversion and Management*, 46(17):2669 – 2688.

[Shensa, 1992] Shensa, M. J. (1992). The discrete wavelet transform: wedding the a trous and mallat algorithms. *IEEE Transactions on signal processing*, 40(10):2464–2482.

[Stolojescu et al., 2010] Stolojescu, C., Railean, I., Moga, S., Lenca, P., and Isar, A. (2010). A wavelet based prediction method for time series. In *Proceedings of Stochastic Modeling Techniques and Data Analysis (SMTDA2010) International Conference, Chania, Greece.*

[Storn and Price, 1997] Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

[Sweldens, 1996] Sweldens, W. (1996). The lifting scheme: A custom-design construction of biorthogonal wavelets. *Applied and computational harmonic analysis*, 3(2):186–200.

[Sweldens, 1998] Sweldens, W. (1998). The lifting scheme: A construction of second generation wavelets. *SIAM Journal on Mathematical Analysis*, 29(2):511–546.

[Vaithiyanathan et al., 2014] Vaithiyanathan, D., Seshasayanan, R., Kunaraj, K., and Keerthiga, J. (2014). An evolved wavelet library based on genetic algorithm. *The Scientific World Journal*, 2014.

[Virtanen et al., 2020] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, İ., Feng, Y., Moore, E. W., Vand erPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and Contributors, S. . . (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

[Wickerhauser, 1994] Wickerhauser, M. V. (1994). *Adapted Wavelet Analysis from Theory to Software*. A. K. Peters, Ltd., USA.

[Wong et al., 2003] Wong, H., Ip, W.-C., Xie, Z., and Lui, X. (2003). Modelling and fore-

casting by wavelets, and the application to exchange rates. *Journal of Applied Statistics*, 30(5):537–553.

[Zhang, 2003] Zhang, G. P. (2003). Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175.

[Zhang et al., 1998] Zhang, P., Patuwo, E., and Hu, M. (1998). Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, 14:35–62.