UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RENATA CORREIA DE ANDRADE

TIME SERIES FORECASTING WITH DEEP FOREST REGRESSION

Recife
2020

RENATA CORREIA DE ANDRADE

TIME SERIES FORECASTING WITH DEEP FOREST REGRESSION

Dissertação de Mestrado apresentada ao programa de pós-graduação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Inteligência Computacional

Orientador: Paulo Salgado Gomes de Mattos Neto

Coorientador: George Darmiton da Cunha Cavalcanti

Recife

2020

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

A553t Andrade, Renata Correia de

Time series forecasting with deep forest regression / Renata Correia de Andrade. – 2020.

83 f.: il., fig., tab.

Orientador: Paulo Salgado Gomes de Mattos Neto.

Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIN, Ciência da Computação, Recife, 2020.

Inclui referências e apêndice.

1. Inteligência computacional. 2. Regressão. 3. Séries temporais. I. Mattos Neto, Paulo Salgado Gomes de (orientador). II. Título.

006.31 CDD (23. ed.) UFPE- CCEN 2021 - 180

Renata Correia de Andrade

"Time Series Forecasting with Deep Forest Regression"

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 30/11/2020.

ACKNOWLEDGEMENTS

This dissertation took over two years to complete. The challenges during this time were numerous: working a full-time job, moving to another state, away from most of my family and friends, and in the last months, a global pandemic. I thank God for all the graces in my life during this troubling time and for the strength to keep pushing forward.

I would not have been able to do this without some very special people, who I also would like to thank.

My parents and my sister, my most enthusiastic supporters. Throughout all my life you have pushed me forward and encouraged me to pursue more. I would not be where I am today without your love, support and understanding.

All my friends who were by my side during this journey. You were responsible for cheering me up on times when I was down and listened without complaint to my constant rants, even when they happened at 2 a.m. at a video call during lockdown.

My advisor, Paulo Salgado, as well as my co-advisor, George Cavalcanti, for believing in me even when I did not believe in myself and for not giving up on me. Thank you for the guidance in this journey.

ABSTRACT

A time series is a collection of ordered observations which are usually measured in repeated intervals. Time series forecasting is an area of research which studies methods for prediction of future values in a series. Forecasting methods range from statistical procedures, such as ARIMA to, more recently, machine learning approaches. Deep neural networks (DNNs) have shown good performance on a great number of tasks, including time series forecasting for which DNNs are considered state-of-the-art. Most deep models today are neural networks, but despite its popularity and proven competitive performance when compared to other machine learning algorithms, DNNs still face some limitations. Most notably, they usually require a large number of training examples - which could be unavailable for smaller time series - and they possess a large number of hyper-parameters which need to be tuned to individual datasets. Multi-grained cascade forest (gcForest) is a deep machine learning algorithm which has been proposed for classification and that addresses DNNs limitations while replicating the features which are responsible for the success of this type of model. This dissertation's goal is to adapt the original gcForest algorithm in order for it to work with regression problems, enabling it to be applied to time series forecasting. The influence of the two different stages of gcForest - multi-grained scanning and cascade forest, is also investigated. Also explored is the possibility of adding an additional model to the end of the cascade forest structure and thus change the way the final result is calculated. Changes to the algorithm are presented and its performance is evaluated on four different time series datasets, according to three performance metrics: mean squared error, mean absolute error and mean absolute percentage error. Results show that gcForest achieves competitive performance on all four datasets, when compared to traditional machine learning models.

Keywords: deep forest; gcForest; regression; time series; time series forecasting.

RESUMO

Uma série temporal é uma sequência de observações medidas em espaços de tempo definidos. Previsão de séries temporais é uma área de pesquisa que estuda métodos para previsão de valores futuros em uma série. Métodos de previsão variam de procedimentos estatísticos, como o ARIMA, para, mais recentemente, abordagens com aprendizagem de máquina. Redes neurais profundas, em específico, mostraram um bom desempenho em uma variedade de problemas, incluindo a previsão de séries temporais, onde são consideradas estado da arte. A maioria dos modelos de aprendizagem profunda atualmente são redes neurais, mas, apesar de sua popularidade e bom desempenho quando comparado a outros algoritmos de aprendizagem de máquina, redes neurais profundas ainda possuem algumas limitações. Mais notavelmente, este tipo de modelo normalmente precisa de uma quantidade maior de exemplos de treinamento, que podem não estar disponíveis para séries temporais mais curtas, além de possuir uma grande quantidade de parâmetros que precisam ser ajustados a cada conjunto de dados. O Multi-grained Cascade Forest (gcForest) é um modelo de aprendizagem profunda proposto para problemas de classificação e que endereça as limitações das redes neurais profundas, enquanto replica características responsáveis pelo sucesso desse tipo de modelo. O objetivo desta dissertação é adaptar o algoritmo original do gcForest para que ele possa ser aplicado a problemas de regressão, possibilitando que o mesmo seja utilizado para previsão de séries temporais. A influência das duas etapas do gcForest - *multi-grained scanning* e *cascade forest* - também é avaliada. Além disso, é explorada a possibilidade de adicionar um regressor ao final da etapa de *cascade forest* e assim alterar a forma de cálculo do resultado final. Após apresentar as mudanças feitas ao algoritmo, seu desempenho é avaliado em quatro séries temporais diferentes de acordo com três métricas de performance: erro médio quadrado, erro médio absoluto e o erro percentual absoluto médio. Resultados mostram que a versão proposta do gcForest atinge um desempenho competitivo quando comparado a modelos tradicionais de aprendizagem de máquina.

Palavras-chave: deep forest; gcForest; regressão; séries temporais; previsão de séries temporais.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ACF	Autocorrelation Function
DBN	Deep Belief Networks
DNN	Deep Neural Network
Extra	Extra Trees Regressor
FNN	Feedforward Neural Network
LR	Linear Regression
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAPE	Mean Percentage Error
MCC	Model Correlation Coefficient
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
RF	Random Forest
RNN	Recurrent Neural Network
SVM	Support Vector Machine
SVR	Support Vector Regressor

CONTENTS

1 INTRODUCTION

This chapter introduces the proposed work, first by giving a context about time series and time series forecasting and then presenting the goals for this dissertation. The last section in the chapter will discuss the dissertation structure for the following chapters.

1.1 CONTEXT

A time series is a sequence of observations measured during a certain interval of time (Chatfield, 2000). Time series forecasting is an area of study that proposes methods to predict future, previously unseen, values in a series.

Time series forecasting techniques are applied in a number of fields. These techniques have been employed, for example, in predicting the number of patients in a hospital's emergency department (Jones et al, 2008), allowing for correct allocation of people and resources. They have also been applied to predict the production of certain species of fish in the Mediterranean sea (Tsitsika, 2007). Another common field in which such techniques are employed is finance - forecasting has been applied to, for example, the prediction of prices of future contracts (Tay and Cao, 2001).

One of the most popular algorithms for time series forecasting is the Autoregressive Integrated Moving Average, or ARIMA. According to (Zhang, 2003), the popularity of ARIMA is due to its statistical properties as well as the Box and Jenkings methodology (Box and Jenkings, 1970). This methodology is a framework for training and forecasting values with ARIMA. In order to apply ARIMA for time series forecasting, two main things are assumed: that the data has linear characteristics and can thus be modelled by a linear model and that the time series is stationary.

An alternative to ARIMA is through modelling the forecasting task as a machine learning problem. Using machine learning has many advantages, including the possibility to model non-linear time series (Zhang, 2003), as well as the fact that conditions on series behavior are unnecessary, such as the necessity for stationarity. Machine learning models, especially neural networks, have been extensively used for time series forecasting, like in (Yona et al, 2008, Nagesh Kumar et al, 2004, Torres et al, 2018). Among this class of models, recurrent neural networks appear to be the algorithm of choice for many of them (Sezer et al, 2020).

According to (Zhou and Feng, 2017), while neural networks are powerful they too still have some limitations. These limitations include, for example, the need for a large amount of training data, which is sometimes unavailable, the large number of parameters that need to be tuned in order for the model to achieve good performance on a specific task and the fact that neural networks work as a “black-boxes”, which makes model explainability difficult. Another important aspect of neural networks mentioned by the authors is that a network architecture must be defined before training, which might lead to an unnecessary increased complexity.

A deep forest algorithm, called Multi-Grained Cascade Forest (or gcForest), was then proposed by (Zhou and Feng, 2017) as an alternative to deep neural networks and was applied to classification problems. It is an ensemble model which consists of two main parts: multi-grained scanning and cascade forest. Multi-grained scanning is responsible for extracting features from datasets where attributes relate to each other, such as is the case with images and sequence data. After this process, the transformed features serve as input to the Cascade Forest step, where the classification prediction is made. The depth of Cascade Forest is automatically defined through the algorithm and the base learners which compose the ensemble are Random Forests and Completely Random Forests, which make explainability easier when compared with neural networks.

Given the advantages of gcForest over neural networks and its results for classification, this dissertation proposes an adaptation of the model in order to work with regression problems, specially time series forecasting. Both stages of cascade forest were adapted to work with time series forecasting and the proposed model also includes the addition of a final regressor after cascade forest, in order to better combine the results of the last layer.

1.2 DISSERTATION GOALS

The main goal of this dissertation is to analyze the performance of deep forest when applied to the task of time series forecasting. In order to achieve this the following milestones are outlined:

Adapt the gcForest classification algorithm to work with time series forecasting. When modelled as a machine learning problem, time series forecasting works as a supervised machine learning task. As gcForest was first proposed for classification, the model needs to be adapted in order to work with regression.

Investigate ways to extract the most out of the algorithm and increase model performance. Because the original algorithm was proposed for classification, it is possible that its original basic configuration might not be adequate for regression. This step, in the context of this study, includes optimizing the model parameters as well as choosing a new method for combining the final model results.

Test the model on different time series datasets. In order to verify if the proposed model is efficient for the task of time series forecasting, its performance must be analysed for more than one dataset.

Compare achieved results with those of other methods of the same type. In possession of the model results, it is important to compare its performance to that of other machine learning algorithms in order to verify if and when it might be a viable option for time series forecasting.

1.3 DISSERTATION STRUCTURE

This dissertation is composed of a total of six chapters, including this introduction.

Chapter 2 will address current popular methods for time series forecasting, including the statistical method ARIMA, as well as three machine learning approaches: deep learning, support vector regression and ensemble learning, all of which will be relevant in the context of the proposed method. All of these methods are accompanied by example cases where they were applied to time series forecasting.

Chapter 3 presents gcForest, a deep forest model first proposed for classification and which achieved great results on several tasks of this type. After introducing the algorithm and broaching, in more detail, the reason why it might be a good fit for time series forecasting, the content turns to the changes that had to be made in order for it to work with regression problems. Further proposed improvements to the model are also presented.

Chapter 4 discusses the methodology used during the experiments. The time series datasets are presented, as well as the data treatment applied to them. Variations of the gcForest models are introduced as well as the optimization strategy used to choose the best model parameters and increase model performance.

Chapter 5 will present the results originated from the experiments described in Chapter 4. A comparison of performance between the complete gcForest model as well as employing only one of its two parts will be provided. Next, the performance for each of the evaluated time series will be presented.

Chapter 6, as the last chapter, presents the dissertation’s conclusions, which are based on the results found in the previous chapter. This chapter also introduces further improvements and lines of research which can be pursued in the future.

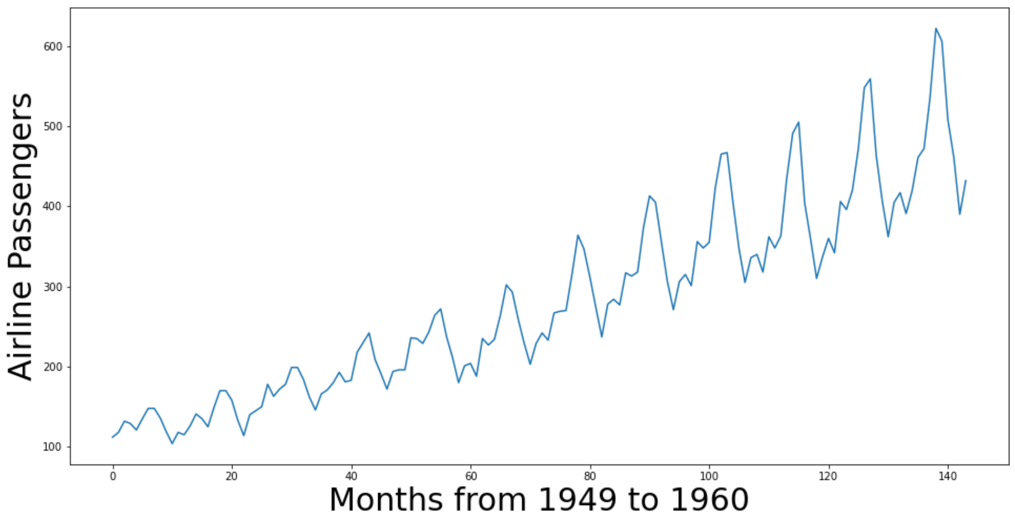
2 TIME SERIES FORECASTING

The first section in the chapter, 2.1, introduces the concept of time series and time series forecasting as well as describes the ARIMA model, one of the most popular approaches to forecasting. The section also explains the motivations behind the usage of machine learning for time series forecasting as an alternative to ARIMA, giving reasons why sometimes it might be preferred to use such an approach. Section 2.2 deals with the first step in the machine learning approach - preparing the data in order for it to serve as input to the models. Sections 2.3, 2.4 and 2.5 address, respectively, three types of learning methods in machine learning for time series forecasting: support vector regression, deep neural networks and ensembles. Finally, section 2.6 presents an overview of the chapter.

2.1 TIME SERIES AND TRADITIONAL FORECASTING MODELS

A time series is a sequence of observations measured during a certain interval of time (Chatfield, 2000). Figure 1 shows an example of a time series representing the total number of United States airline passengers. Observations were measured monthly from 1949 to 1960 and are of order 10^3.

Figure 1 - Number of United States airline passengers.



Source: R Core Team, 2020.

Forecasting in the field of time series means predicting future values in a sequence, based on past observations. The reason why forecasting might be useful varies depending on the situation. In the case of the airline time series, estimating the number of passengers for the following month might help airlines to better prepare by, for example, making sure that enough aircrafts and staff are available to address the demand or to plan financially in the case where fewer tickets are predicted to be sold.

A popular algorithm for time series forecasting is the Autoregressive Integrated Moving Average, or ARIMA. The concept behind this model is that a future value can be modelled through a linear combination of past values and random errors (Zhang, 2003).

One particular aspect of ARIMA is the restriction that the time series data be stationary. A stationary time series is one for which statistical properties, such as the mean, remain the same throughout the series. Identifying stationarity in the data is one of the first steps in the Box and Jenkins method (Box and Jenkings, 1970), which comprises a series of steps for training and forecasting values with ARIMA. The ARIMA algorithm itself can turn a series stationary through a differencing procedure. Through this procedure, future values of a time series, such as the one presented in Figure 1, which has a trend and as such is not stationary, can be predicted using ARIMA.

Statistical approaches to time series forecasting, such as ARIMA, while widely popular, are limited in the way that they rely on the assumption that the data can be modelled by a linear function and thus cannot model non-linear characteristics, even if, in some cases, treating the data as linear might not apply to real-world time series (Zhang, 2003). One way to model non-linear characteristics in the data is by using a machine learning approach and the first step in this direction is the modelling of the time series data in a way that it can serve as input to a machine learning model.

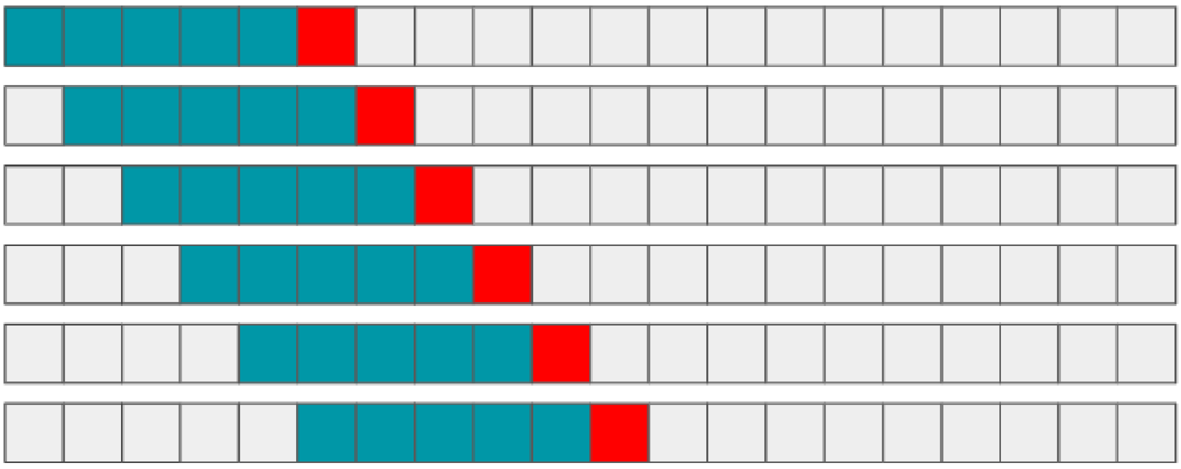
2.2 MODELLING THE DATA AS A SUPERVISED MACHINE LEARNING PROBLEM

In order for a time series dataset to be used as input to a machine learning model, it first needs to be adapted as such. One way to achieve this is by using a sliding window. Figure 2 illustrates the sliding window process. This process consists of using *n* previous consecutive observations (called *lags*) to estimate one-step ahead. The optimal value of *n* can be different for distinct time series and it is possible to include *n* as one of the parameters during model optimization processes. After the window is generated, it is also possible to filter which lags will effectively be used during training, through methods such as the autocorrelation function (ACF).

By making use of a sliding window, a time series dataset is modelled as a supervised learning problem. In supervised learning, the output value - which can be discrete and limited, in the case of classification, or continuous, in the case of regression - is known to the model for the training dataset.

Because the output value is known, it is possible to, more directly, measure the performance of a model trained on this data, as a test set can be labeled. Another advantage of the sliding window methods is that it allows the transformed time series to be modelled by any available machine learning algorithm for regression (Dietterich, 2002).

Figure 2 - Illustration of the process of applying a sliding window to a time series dataset.



Source: Author, 2020. In this case, n=5 and data points in read are modelled as a result of the 5 previous observations.

2.3 DEEP NEURAL NETWORKS FOR TIME SERIES FORECASTING

Deep learning is an area of machine learning that gained its popularity after achieving superior performance on tasks which deal with image and audio (Zhou and Feng, 2019). Since then, deep learning has been employed in a number of fields, such as finance (Oncharoen and Vateekul, 2018), genetics (Quang et al, 2014) and agriculture (Li et al, 2016).

Today, most deep learning models are based on neural networks (Zhou and Feng, 2019). The type of neural networks employed vary according to the task being performed, with the most common approach involving a multi-layer perceptron (MLP) (Goodfellow et al, 2016), a type of feedforward network, or FNN. Other kinds of deep neural networks (DNNs) have different architecture and are specially suited to solving specific tasks, such as convolutional neural networks (CNNs) (Lecun, 1989) for problems involving images and Recurrent Neural Networks (RNNs) (Rumelhart et al, 1986) for dealing with sequence data.

Given the popularity of the deep feedforward network and the suitability of RNNs for sequence data, such as time series, these two types of DNNs will be further discussed in this chapter.

2.3.1 Feedforward Networks

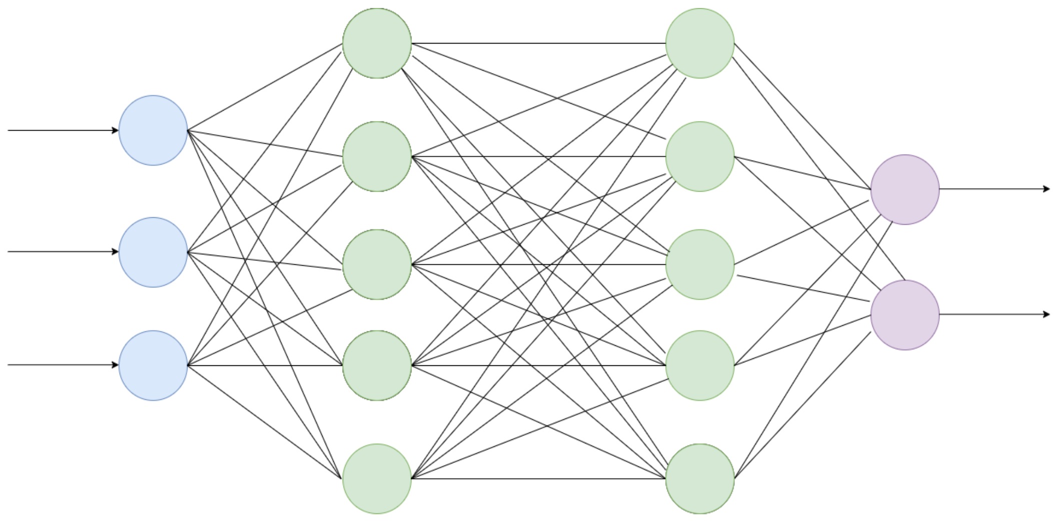
Feedforward networks, also called Multi-Layer Perceptrons (MLP), are the most common type of neural networks (Goodfellow et al, 2016). They are also the most common choice of neural networks when dealing with time series forecasting problems (Torres et al, 2018).

The aim of an MLP is to approximate a function f that maps an input x to an output y, such as that $y = f(x)$. The basic structure of FNNs is the neuron, which grouped together, form a layer. MLPs are composed mainly of three types of artificial layers: an input and output layer, between which lie the hidden layers. One or more hidden layers can be present in the model, with the name deep learning referring to a network composed of multiple hidden layers (Goodfellow et al, 2016).

Neurons in all layers are connected by weights, which are used to map the inputs into the outputs. These weights are updated through a process called *backpropagation*, by which the error produced by an output is propagated backwards into the network.

Figure 3 illustrates a feedforward neural network, which has its name because information flows only in one direction, in this case from left to right or from input to output. The image represents an MLP with two hidden layers and five neurons in each of them, as well as input and output layers with three and two neurons, respectively.

Figure 3 - Illustration of a feedforward neural network.



Source: Author, 2020. Input layer neurons are represented in blue, while those in the hidden and output layer are colored green and purple, respectively. The arrows represent the direction of the information flow.

The authors in (Torres et al, 2018) used a deep feedforward network in order to solve a time series forecasting problem, which in this case was related to the electricity consumption in Spain. The proposed model aimed to forecast values in a big data time series by using deep learning. In order to address problems related to the large number of instances in this time series, a distributed computing framework is introduced. Results show that the proposed deep learning model performs better than traditional algorithms such as Random Forests and Gradient-Boosted Trees.

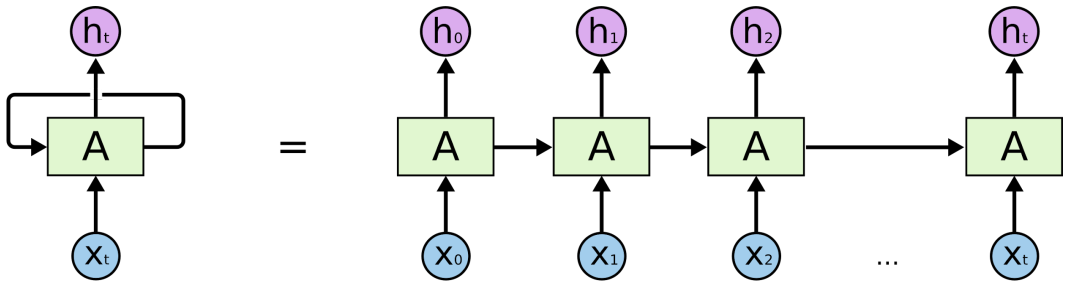
A FNN is employed by (Yona et al, 2008) in order to predict insolation and forecast the power generated through photovoltaic systems. The performance of this FNN however, is lower than that of a recurrent neural network.

2.3.2 Recurrent Neural Networks

Recurrent neural networks (Rumelhart et al, 1986), or RNNs, are a type of neural network which are specially suited for dealing with sequence data, such as time series. In fact, a recent study on financial time series forecasting (Sezer et al, 2020) showed that in this specific field, RNNs were the model of choice in the majority of the papers published from 2005 to 2019, which used deep learning models.

A RNN differs from feedforward neural networks in the sense that not only the current input is used in order to produce an output, but inputs are combined with information previously seen by the network as a form of feedback. According to (Goodfellow et al, 2016), the type of feedback, or recurrency, might vary, being, for example, providing this feedback after a single time step or only after the complete sequence is read by the network. Figure 4 illustrates such a recurrent neural network, where A is the core of the neural network, and *x(t)* and *h(t)* are the models input and output at time *t*, respectively.

Figure 4 - Recurrent neural network structure, representing the information flow, where the output is fed back as input.



Source: Olah, 2015.

Just like in (Yona et al, 2008), the authors of (Nagesh Kumar et al, 2004) also compare the performance of an RNN to that of a FNN. A partial recurrent model is employed to predict monthly river flow in India, and its performance is also superior to that of the FNN.

A Long-Short Term Memory network, or LSTM (Hochreiter and Schmidhuber, 1997), is a type of recurrent neural network. They were proposed as an alternative to traditional RNNs for problems where the value being predicted is distant from the most recent information in the sequence, as traditional RNNs do not seem to be able to deal with these dependencies without careful parameter tuning (Olah, 2015).

The authors in (Siami-Namini et al, 2018) performed an empirical study in which the ARIMA model was compared to a LSTM. During the study, twelve financial time series were used to train both models and their performance was compared using a root-mean-square error. For all twelve time series, the LSTM model outperformed ARIMA by a large margin, with an error reduction rate varying between 84% and 87%.

Recurrent neural networks have even been employed in predicting transmission of COVID-19 cases during the 2020 global pandemic. The authors in (Chimmula and Zhang, 2020) propose using a LSTM to forecast the transmission of this disease in Canada.

2.4 SUPPORT VECTOR REGRESSION FOR TIME SERIES FORECASTING.

A popular choice when it comes to solving machine learning problems is the Support Vector Machine, or SVM. The regression version of the SVM is often referred to as SVR, or Support Vector Regression.

The SVM can be used to model both linear and nonlinear functions, with the modelling of non-linear outputs being achieved by mapping the original input features into a higher dimension feature space and finding a linear solution in this new space (Müller and Guido, 2016). The efficient mapping of the features is done by using a kernel function, which can be chosen according to the dataset in order to achieve the best performance. Advantages of the SVR include the fact that the algorithm does not suffer from problems with local minimums, which reduces the chances of overfitting (Kim, 2003). This characteristic, as well as the fact that the SVR has fewer parameters to be optimized, are reasons why, in some cases, an SVR might be a better choice than neural networks (Tay and Cao, 2001).

The process of training an SVR involves minimizing a regularized error function, which is composed by an ϵ-insensitive error (Bishop, 2006). An ϵ-insensitive error function is one in which the error is considered to be zero in the case when the absolute difference between true and predicted values is less than ϵ (Bishop, 2006). Thus, the objective is to find a function that maps the output in a way that this absolute difference is always less than a predefined value ϵ.

In (Tay and Cao, 2001), an SVM is used in the forecasting of five financial time series. The datasets are modelled not by using a sliding window, but instead by creating five custom features, four of which are lagged values corresponding to 5-day periods and one is a transformed feature obtained through a 15-day moving average. For all five time series, the SVM model performed better than a neural network trained through back-propagation.

The authors in (Chen and Wang, 2005) combine an SVR with genetic algorithms in order to predict tourist demand. The genetic algorithm is used to optimize the SVR parameters. This combination of models showed a better performance than both a back-propagation neural network and ARIMA when predicting the quarterly number of visitors to China.

2.5 ENSEMBLES FOR TIME SERIES FORECASTING

An ensemble model is one composed of more than one base algorithm, which are combined in order to decide the final output (Zhou, 2012). The base algorithm can be any available model which can be applied to the problem being solved. Ensembles can be either *homogeneous*, when composed by variants of the same type of base algorithm, or *heterogeneous*, when base models are of a different type.

In order to compose an ensemble, a number of strategies can be employed. These strategies are mainly based on alterations made to the data and/or to the base models. Such strategies are discussed in (Zhou, 2012) and recounted briefly below - they can be employed both individually as well as together.

- **Changes in the Data.** This approach of creating an ensemble includes training the base models on variations of the data for the problem being solved. Alternate datasets can be generated by, for example, randomly subsampling the training data and fitting each base learner to a reduced version of the original dataset - this process is called *bagging.* Another approach, referred to as *boosting,* consists of training different model sequentially, where the next model to be trained specializes on the training samples that the previously trained model failed to correctly predict;
- **Changes in the Base Models**. One approach to building an ensemble is making changes to the base that compose it. These changes could be in the form of different types of base models, as well as in the introduction of different values for hyper-parameters of learners of the same type.

After training each individual base learner in the ensemble, the next step consists in combining their result in order to achieve a single output value. Strategies for combining results include averaging, voting, and a combination by learning method. All three of the previously mentioned strategies are described below, also as described by (Zhou, 2012).

- **Average**. There are two main ways to average the results for the models in an ensemble. The first one is a direct calculation of the average by adding the results and then dividing by the total number of learners in the ensemble. The second approach is a weighted average, where each individual model is assigned a weight, which will bestow a level of importance to the learner's result.
- **Voting.** This is a combination approach most fitted to classification problems - being the most commonly used method in this group of tasks. When voting is employed, each model in the ensemble chooses a class to put the sample in. The most used form of voting, called

majority voting, simply selects the class with the most votes as the final answer.

- **Combining by learner.** Out of the three combining methods presented in this topic, this is the most complex, as it involves training another learner in order to combine the outputs in the ensemble. The training data for this final learner will be the outputs from the other ensemble models, while its output will be the final ensemble result.

2.5.1 Diversity in Ensembles

Even without a formal, widely accepted definition of diversity, it is a consensus that is an important concept when planning a model ensemble (Zhou, 2012). The ensemble performance depends on the base learners that compose it being diverse, as well as in a combination of strong and weak models, in such a way that they complement each other (Zhou and Feng, 2019).

The two strategies previously discussed as an approach to ensemble building - changes in the data and changes in the base base model - can also be viewed as diversity enhancement techniques, as diversity itself can be viewed as the act of introducing random aspects during model building (Zhou and Feng, 2019).

Other strategies for increased diversity are discussed by (Zhou and Feng, 2019). These strategies include altering the input features, for example by randomly selecting a subset of features to train each model, as well as altering output results by using strategies such as output flipping for classification (Breiman, 2000) or output smearing for regression (Breiman, 2000).

2.5.2 Ensembles in Time Series Forecasting

Employing ensembles in solving time series forecasting tasks has been done a few times in literature (Oliveira and Torgo, 2014; Qiu et al, 2014; Adhikari et al, 2015), while the exact strategy varies according to the study in question to various degrees of successful results.

The authors in (Qiu et al, 2014), for example, employ a combination of deep belief networks (DBNs), in what is one of the first studies to use an ensemble of deep models for time series forecasting. Diversity between the models in the ensemble is generated by altering the maximum number of epochs for which a DBN can be trained. The model results use a combination by learning strategy - a support vector machine for regression is used as the combination model.

Another approach which uses neural networks in an ensemble for time series forecasting is the one presented in (Adhikari et al, 2015). Along with neural networks, other machine learning as well as statistical models such as ARIMA are also included in the ensemble. Combining linear models, such as ARIMA, with non-linear ones, such as neural networks might contribute to increased model performance, as different patterns in the data can be modelled by individual algorithms (Zhang, 2003).

Diversity in ensembles for time series forecasting is not achieved only by changing the actual learning models, like in the two previous studies discussed above, but also by altering the modelling of the time series data. The authors in (Oliveira and Torgo, 2014) evaluate the impact of different diversity inducing methods on the training of regression trees with bagging. In this study, strategies to increase diversity include training models with instances generated from different sizes of sliding windows, as well as complementing the inputs by adding summary statistics of recent values, like the mean and variance.

2.6 CHAPTER OVERVIEW

The chapter first introduced the concept of time series and time series forecasting while discussing one of the most common methods for predicting future values in a series: ARIMA.

As an alternative to ARIMA, this chapter also presented an overview of the support vector regression and of neural networks, especially deep neural networks. While feedforward neural networks are the most basic example in this class of machine learning algorithms, other deep models have better performance in certains fields, like convolutional neural networks for image and recurrent neural networks for sequence data, such as time series. Both FNNs as well as RNNs have been employed in time series forecasting problems, with RNNs outperforming FNNs in at least one example.

The chapter also broached the subject of ensembles in machine learning. Ensembles is another approach which has been employed to time series forecasting. For this specific kind of task, most ensemble base learners are deep learning models, neural networks in particular.

In fact, neural networks have been the base model for machine learning algorithms for some time, either being used by themselves or in combinations. Next chapter will introduce an alternative to deep neural networks and explain why such a model might be a good idea for time series forecasting.

3 THE PROPOSED APPROACH

As described in the last chapter, machine learning models have proved to be a viable alternative for time series forecasting tasks. This dissertation employs a machine learning approach for time series forecasting which is based on a model that has shown competitive results on classification problems - gcForest[1]. This chapter will explain how gcForest works as well as the adaptations made to the model in order for it to be used in regression tasks.

Section 3.1 will discuss the reasons why gcForest is a promising machine learning model and the inspirations behind its creation, many of which also apply to regression and time series forecasting problems, the reason it was chosen for this study. Section 3.2 describes the gcForest classification algorithm itself, while Section 3.3 will elaborate on the changes made for it to work with regression. One of the two stages of gcForest, multi-grained scanning, might have a significant impact on time series forecasting and will be further discussed in Section 3.3. The following section, 3.4, will introduce other proposed changes to the algorithm, which aim to increase its performance. The last section, 3.5, presents an overview of the chapter.

3.1 GCFOREST

Neural networks, in particular deep neural networks (DNNs), have shown superior performance to other machine learning algorithms in a number of tasks, which range from image and text classification (Krizhevsky et al, 2012; Conneau et al, 2017), machine translation (Wu et al, 2016), even time series forecasting (Qiu et al, 2014).

On the other hand, the training of deep neural models represents certain challenges that are particular to this group of algorithms. They require a large number of training instances in order to achieve competitive performance and they possess a large number of hyper-parameters that need to be tuned so that the model can adapt itself to the problem in question (Zhou and Feng, 2017). The authors in (Zhou and Feng, 2017) also point out that the architecture of a deep neural network

[1] Pronounced ‘Geek Forest’, as instructed by the original authors.

has to be established before the training process begin, which makes the definition of model complexity independent from the data used in training - this can lead to overly complex models, while those which are simpler might obtain better performance.

It is also noted in the paper that the decision process of DNNs is hard to explain. Model interpretability in machine learning is an active area of research, with the reasons behind the appeal of interpretability ranging from the desire to understand causality in the data to the assurance that the model is making “fair and ethical” decisions (Lipton, 2018).

After discussing the issues with deep neural networks, the authors in (Zhou and Feng, 2017) proposed a new deep learning model that is not composed of neural networks: gcForest. This new algorithm is an ensemble model and possesses three characteristics which the authors believe to be the key behind DNNs success: layer-by-layer learning, feature transformations during model training and model complexity. These characteristics are further explained in an expansion of the original paper (Zhou and Feng, 2019) which also details the reasons behind their beliefs.

The proposed model, gcForest, achieved competitive performance on several classification tasks. It performed better than deep neural networks models on classical machine learning datasets such as MNIST (LeCun et al, 1998) and IMDB (Maas et al, 2011). gcForest also performed better than neural networks on UCI datasets (Lichman, 2013) with a low number of features. It is important to note that all results reported in the article were generated using a single configuration of parameters for gcForest - which shows the model can achieve good results without the need for tuning, even if the authors mention that even better results could be achieved by adjusting these parameters to the problem.

Since deep learning models constructed with neural networks usually need a large number of examples to train a model, an approach which can achieve competitive performance with fewer samples would be specially fitting for time series forecasting as some time series can be composed of only dozens or hundreds observations. The gcForest algorithm addresses that by employing base models which do not require so many training samples as a neural network.

There is no single model which works best with all time series (Zhang, 2003), the best choice will depend on the dataset. Apart from choosing the model themselves, often it is still necessary to fine tune the algorithm in order for it to work with a certain dataset. Not needing to fine-tune for parameters would also be convenient as a single model configuration could be used for different time-series datasets - gcForest achieves this feat for several classification problems.

As discussed in the previous chapter, machine learning and deep learning models have shown promise when applied to solving time series forecasting problems. Gaining the benefits provided by DNNs while also circumventing some of its problems makes gcForest an appealing candidate for problem solving.

3.2 THE GCFOREST ALGORITHM

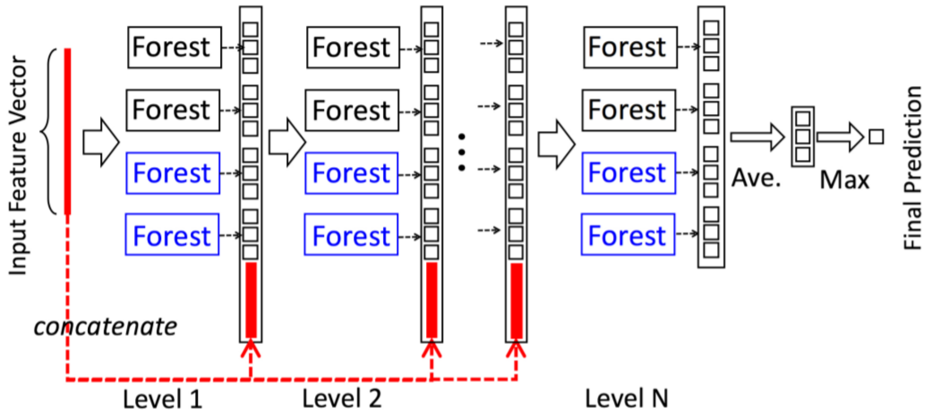
gcForest (Zhou and Feng, 2017) was proposed as an alternative to deep neural networks. As such, the model's algorithm was conceived in a way that mimicked some of the DNN's features. This algorithm can be divided into two separate sections.

The main part of the model, called Cascade Forest, is a form of ensemble. In machine learning, an ensemble is a combination of models which complement each other in order to achieve performances that are better than that of a single model (Zhou, 2012). The second part of gcForest is called Multi-Grained Scanning and is a form of feature extraction that aims to pre-process the input data when the data in question has some kind of spatial relation between its features. This is the case, for example, for images, where pixel values are related to other pixels around it, or for text, if it is modelled as a sequence of words. Sections 3.2.1 and 3.2.2 will describe each of these two stages in detail as well as how they connect to each other.

3.2.1 Cascade Forest

The Cascade Forest stage of the gcForest algorithm is where the actual predictions are made and thus constitutes the main part of gcForest. The inspiration behind the cascade forest structure was the representation learning in deep neural networks, which occur from layer to layer (Zhou and Feng, 2019). Figure 5, from the original article (Zhou and Feng, 2017), shows the cascade forest structure in detail.

Figure 5 - Cascade Forest structure.



Source: Zhou and Feng, 2017. In the example each cascade layer is made up of four classifiers. The output of each layer is concatenated to the original input features and fed to the following layer.

The dataset features serve as input to the first layer of the cascade structure. A layer is composed of one or several classifiers - the presence of different classifiers in a layer is encouraged in order to increase diversity in the model. The output of a layer, combined with the same dataset features will be the input to the following layer.

All classifiers in a level will have as output a class probability vector, with a length that is the same as the number of classes in the classification problem the model is being trained to solve. Each position in this vector will represent the probability of an instance belonging to a certain class.

In order to define the final output of a layer, a *k*-fold cross validation strategy can be employed. The final class probability vector will be the averaged class probabilities from all the *k* folds. The cross validation was introduced in an attempt to avoid overfitting, which is common in stacked models (Zhou and Feng, 2017; Zhou and Feng, 2019).

Each model in the layer will produce a class vector, which will be concatenated to the input samples and serve as training data to the next layer. As an example, if the cascade structure is being trained on a classification dataset with 10 classes and each layer is composed of 2 different classifiers, from the second layer onwards, instances in the dataset will have 20 additional features - 10 for each classifier.

After a new layer is included in the cascade structure, the model is evaluated on a validation dataset if one is provided, or on the training data itself, in the absence of a validation set. If there is no improvement in accuracy, the training will stop and the final number of layers will be defined.

In order to generate the final classification results, class probability vectors from models in the last layers are averaged and a final class vector is generated. The class of and data instance will correspond to the position in the vector with higher probability.

3.2.2 Multi-Grained Scanning

According to the authors in (Zhou and Feng, 2017), the multi-grained scanning procedure was inspired by the capacity of deep neural networks to extract feature relationships in the data and was introduced in order to increase the performance of the cascade forest stage.The procedure consists of scanning the original features with sliding windows and generating new training instances, which will be used to train classifiers that will produce transformed features. These new features will then be used to train the cascade forest model.

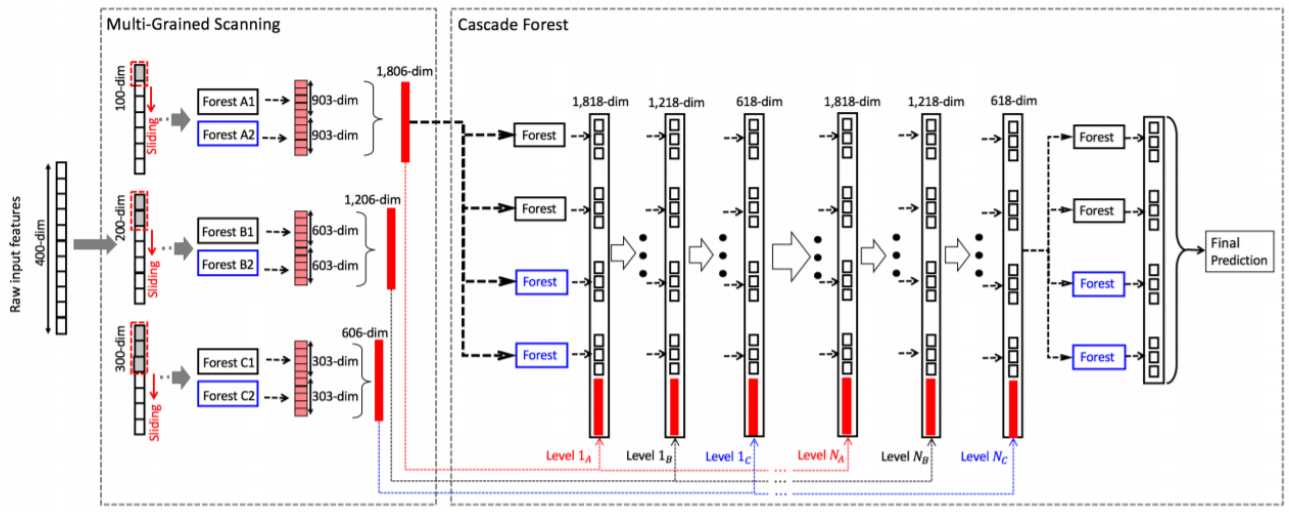
The sliding window can be either one or two dimensional, depending on the type of original input data. An image, for example, will have a two dimensional window while sequence data will have a one dimensional window. The actual size of the sliding window is one of the gcForest parameters. The sliding window scan will generate new instances from the original configuration of features and the class of these new instances will be the same class of the original. In a binary classification, with sequence data, where a single positive sample has 100 features and a sliding window of size 10 is used, 91 new samples will be created and all of them will belong to the positive class.

After these new instances are generated, they are used to train a random forest and a completely random forest. These two models will generate class probability vectors for each instance. The class vectors are then concatenated in order to serve as input to the cascade forest structure. In the binary classification example, the sample with 100 features would be transformed into a new sample with 364 features.

Figure 6, from the original article (Zhou and Feng, 2017), illustrates the complete gcForest procedure. It considers as input a sequence-like dataset, with 400 features in each instance. Three different sliding windows are used to scan the data and train the feature extractor models. Two such models are used and a problem with three classes is considered. This will result in transformed feature vectors of sizes 1806, 1206 and 606 for sliding windows of sizes 100, 200 and 300, respectively.

Because more than one sliding window was used, resulting in samples with a different number of features, each new feature vector will be used to train a level of the cascade structure. This way, a layer is composed not only of a single level, but of *n* levels, where *n* is the number of sliding windows.

Figure 6 - The overall procedure of gcForest, with the multi-grained scanning phase providing the input features for the cascade forest structure.



Source: Zhou and Feng, 2017.

3.3 GCFOREST REGRESSOR

The original gcForest model was proposed as an algorithm to solve classification problems. As discussed in the previous chapter, time series forecasting through machine learning techniques can be achieved by modelling the sequence as a regression problem. So in order to test the performance of gcForest, it first has to be adapted for regression and specifically, time series forecasting. Changes made to the cascade forest will be discussed first while the multi-grained scanning phase will be addressed in the following sections.

While versions of deep forest have been proposed for regression, there is still no widely accepted definition. An example is the work of (Wu et al, 2018), where fcXGBoost was proposed. fcXGBoost is a model based on gcForest that uses eXtreme Gradient Boosting as the base learner and is applied to the task for estimating a person's age from an image. In this model, multi-grained scanning was replaced by a procedure called Multi-Feature Fusion. A version of gcForest for regression was employed in the prediction of time of failure for an automobile (Chen et al, 2020) based on maintenance data - in this version, random forests for regression were used as base models, while a model correlation coefficient (MCC) was used in the place of accuracy for optimization. Deep forest regression has also been employed to short-term load forecasting (Yin et al, 2020), where data from previous days combined with meteorological information was used to predict the wanted value.

As the original gcForest model allows for a high degree of flexibility, the exact form of the model might depend on the problem being solved as well as on the nature and characteristics of the input data. The version proposed in this dissertation aims to be specifically applied to time series forecasting. Certain decisions, for example which metric to use for optimization and how to deal with the k-fold cross-validation step were based on this, as will be explained in the following sections.

3.3.1 Cascade Forest for Regression and Time Series Forecasting

The first step in the adaptation of the cascade forest structure is changing the base models which can compose the basic layer architecture. Instead of classifiers, regressors need to be used in the composition.

By changing the type of base model, the type of model output is also altered. Instead of a class probability vector, the output of a regressor will be the actual predicted value. This behaviour will change the way the final result in the prediction is obtained. Instead of a discrete output, represented by the position with highest probability in the averaged class vector of the last layer, the final result will be a continuous value, extracted by adding and averaging the outputs of the regressors in the last layer of the cascade forest structure.

In order to automatically decide the total number of layers during training, the prediction on the data is evaluated and model performance has to increase after the addition of a new layer. The metric used to measure performance in the original cascade forest is accuracy. In the classification context, accuracy represents the percentage of instances which were correctly predicted. If applied to regression, this would mean the model predictions would have to be an exact match to the truth value in order for the instance to be considered correctly predicted. Instead, regression performance is usually measured by metrics which represent the difference between predicted and truth values - how this difference is calculated varies from metric to metric.

It is important to alter the metric used to verify if the addition of a new layer to the cascade structure is reflected in an increase of overall model performance. The metric chosen was the mean-squared error (MSE). This is one of the most common metrics used in the assessment of performance for time series forecasting (De Mattos Neto et al, 2014), being one the performance measures of choice for several authors (Silva et al, 2018; De O. Santos Júnior et al, 2019; Zhang, 2003). Equation 1 describes how the MSE is calculated. For this equation, *t(i)* is the true value in the series at time *i,* while *p(i)* is the predicted value at time *i* and *n* is the number of samples.

Different regression metrics could be chosen for this step, to the point where the metric could be one of the fitness functions in the optimization process of the model.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left(t\left(i\right) - p\left(i\right)\right)^2 \tag{1}$$

In the original gcForest model, a *k*-fold cross validation is applied when defining the final performance of a layer. This kind of validation strategy can be directly replicated for most regression problems, but that is not the case for time series forecasting. A time series is a sequence of observations and their order needs to be respected. By randomly choosing which instances will belong to a fold, the model could be trained on a dataset that contains future observations, while being evaluated on instances belonging to the past. So when training a gcForest Regressor model with time series data, the cross validation step needs to be omitted.

3.3.2 Multi-Grained Scanning for Regression and Time Series Forecasting

Multi-grained scanning is a step designed in order to extract transformed features from the original input vector in the cases where those features relate to each other, as is the case with time series forecasting. As such, a time series forecasting task could benefit from a strategy like multi-grained scanning.

The adaptation process for multi-grained scanning in order for it to be used in regression has fewer steps than that of the cascade forest structure. The only direct modification necessary would be to replace the classifiers used in this stage with regression models. All other steps, including the scanning with the sliding window and concatenation of the features after transformation can be done with modifications.

That being said, it is important to note that the inspiration for the design of this procedure was drawn from techniques that directly apply to classification and not necessarily regression, which is not the case with the cascade structure.

It is mentioned in (Zhou and Feng, 2017) that by the sliding window scan and the attribution of the original label to the new instances might result in the addition of incorrectly classified samples. This behaviour is related to a method developed to increase diversity in ensembles, called flipping output.

The flipping output method was first introduced by (Breiman, 2000) as a diversity enhancement method for classification. The same article proposes a different method for regression: output smearing. Out Smearing consists not of introducing instances with a wrong label, but in slightly altering the truth value of a sample by adding Gaussian noise to some of those samples.

Another possible issue would be the nature of the generated samples themselves. In order to model a time series a supervised learning task, a sliding window, similar to that used during multi-grained scanning can be used - this means that samples will share features with instances that come before and after them in a sequence, even if these features are in different positions. By scanning all samples with a second sliding window, the result will be that the exact same features, occupying the same position in the vector, will be mapped to different truth values.

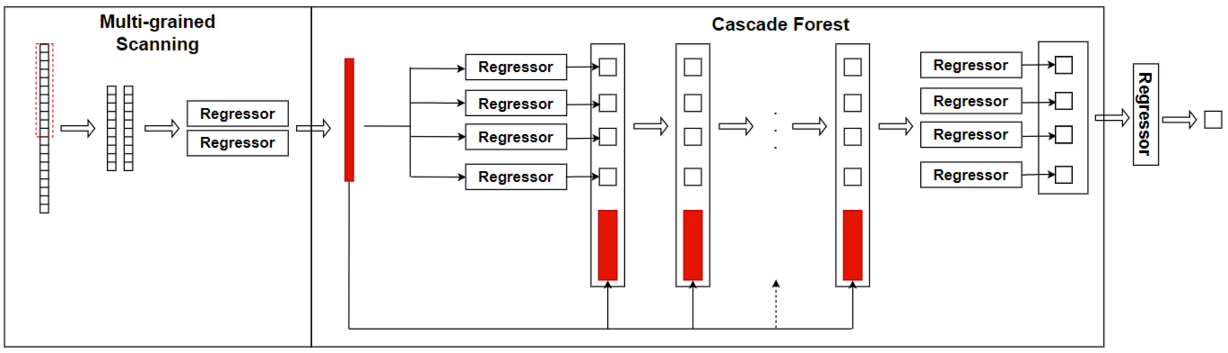
For the reasons presented above, it is a hypothesis raised by this study that the multi-grained scanning stage, as it was designed, will not increase the performance of the cascade forest for time series forecasting. As such, experiments will be conducted in order to assert this hypothesis.

3.4 FURTHER IMPROVEMENTS TO THE PROPOSED MODEL

Cascade Forest is an ensemble algorithm which averages the results from the classifiers on the last layer in order to obtain the final model result (Zhou and Feng, 2017). The regression version also uses the average as a means of combining the results from the regressors in the last layer. In fact, averaging is one of the most common forms of result combination in ensembles because of its simplicity (Zhou, 2012).

Another form of combining the results in an ensemble is by using another learner model, which will have as input the outputs from the other models in the ensemble - this type of combination is called stacking (Zhou, 2012). The structure of gcForest also relates to stacking (Zhou and Feng, 2017), since the outputs from a layer are used as inputs for the following layer.

Figure 7 - Structure of the altered gcForest model.



Source: Author, 2020. Regressors are used in the place of classifiers and the model output is a direct prediction instead of a class probability vector. As a way of combining the outputs in the last layer, a single model is added.

In hopes of improving model performance and inspired by stacking, a method that works for regression (Breiman, 1996), the addition of a final model is proposed for the model. After gcForest is trained by itself, the training samples are used as input and the outputs of the final cascade forest layer are employed in the creation of a new dataset, which in turn will serve as training data for the final model in the cascade structure. This final model’s output will be the actual predicted value. Figure 7 illustrates the final model, composed by the proposed changes, including the addition of the final regressor.

3.5 CHAPTER OVERVIEW

This chapter introduced gcForest - a deep machine learning model that is not composed of neural networks. The model aims to mimic the characteristics of DNNs that make them successful, while addressing some of its problems.

Such problems include the need for a larger number of training samples, when compared to other machine learning models, as well as the need to fine tune the model parameters in order to achieve good performance. A lack of training samples is a common problem in time series forecasting, as a time series can be composed of a restricted number of instances. Combining the output of different models has also been shown to improve performance of time series forecasting tasks. These points indicate that gcForest might be a good fit for forecasting.

The gcForet model was conceived for classification. For it to work with regression some alterations had to be made. These alterations were discussed in this chapter, along with the possibility of adding an additional model to the end of the cascade structure.

4 EXPERIMENTS METHODOLOGY

This chapter aims to explain the methodology used in the experiments. The first topic will discuss the time series datasets used as benchmarks, as well as the transformations applied to each one of the series in order for them to be used in the training of the machine learning forecasting models. The second topic, on the other hand, discusses the gcForest Regressor model along with the strategy used for the it's optimization and the definition of its basic architecture, as well as the traditional machine learning algorithms used as a comparison for the proposed model. The third section addresses the metrics used to evaluate said models. The following topic talks about the implementation of the models as well as the machine the experiments were run on. In Section 4.5, an overview of the chapter is presented.

4.1 DATA PREPARATION

In order to test the performance of the proposed algorithm and its variations, as well as to enable their comparison to already established machine learning algorithms, all models were trained and evaluated on four different datasets.

The Canadian Lynx time series dataset consists of annual observations (from 1821 to 1934) of the number of lynxes found by the Mackenzie River district in Canada. The Star time series data set is composed of 600 observations representing the brightness magnitude of a star and was measured daily on consecutive nights. The third time series, referred to as Sunspot, consists of annual measures of spots found during observations of the sun, measured from 1700 to 1987. A financial time series, S&P 500, is a set of observations which represent the monthly market close value from 1970 until 2003. Table 1 describes these datasets in terms of number of observations in each time series along with the maximum and minimum value in each set.

As a way of preparing the data to be used in the training and testing of the models, all datasets were normalized between 0 and 1. For the Lynx dataset, the base 10 logarithm was also extracted from each data point before the normalization process, in order to enable future comparisons of the results to other studies in which this dataset was used and this transformation was also applied, like in (Silva et al, 2018).

Table 1 - Description of the datasets used during the experiments phase.

Dataset	Number of Observations	Maximum Value	Minimum Value
Star Brightness	600	34	0
Sunspot	289	190.2	0
Canadian Lynx	114	6991	39
S&P 500	369	8.423626	4.305739

Source: Author, 2020. The table also includes the total number of observations in each time series as well as the maximum and minimum values, which were used for normalization.

In order to use the data of each dataset as input to the machine learning algorithms, it is necessary to model the observations in a way that each point in the series is a possible target and previous points are used as features - attributes used by the model in order to predict the target value.

As a way to achieve this scenario, a sliding window was run through the data. The maximum window size was set to 20 - this means that in order to predict a single data point in the series, a model could use up to 20 previous observations. The final window size was one of the parameters optimized for each machine learning algorithm.

With a maximum window size of 20, this means that the first twenty observations in the series were not used as target values but only as features for future data points. This was done in order to guarantee that all models were trained and evaluated on the same exact observations and performance could be better compared.

The next step in the data preparation process was dividing each dataset into three smaller ones. The first and largest one, consisting of 50% of the data, is the training dataset, which was used in the optimization stage of the experiment. During this stage, a second dataset is also used - this one consisting of 25% of the time series observations. This second dataset is used as validation data and will serve as an evaluation dataset during optimization. The third set and last observations, also consisting of 25% of the data, was used for testing, after the best combination of parameters was found. Table 2 shows the size of each dataset for all four time series.

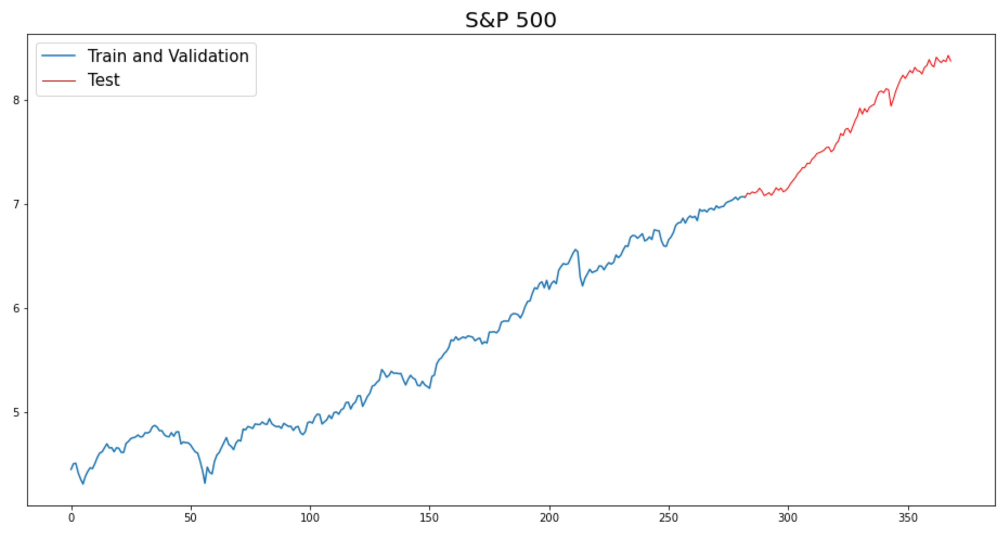
Because a time series is a sequence of ordered observations, this order was preserved when defining the three sets of data. This training set consists of the first 50% of data points while validation includes the next 25% and the test set the last 25%. Figure 8 illustrates this scenario for the S&P 500 time series, with data points used for testing highlighted in red and those employed during training and validation represented in blue.

Table 2 - Distribution of observations after defining a sliding window of 20.

Dataset	Number of Observations	Number of Training Samples	Number of Validation Samples	Number of Test Samples
Star Brightness	600	290	145	145
Sunspot	289	135	67	67
Canadian Lynx	114	47	23	24
S&P 500	369	175	87	87

Source: Author, 2020. The training dataset corresponds to 50% of the data, while the validation and test datasets, 25% each.

Figure 8 - Plot of the S&P 500 time series, with data points used for testing highlighted in red.



Source: Author, 2020.

4.2 MODEL AND PARAMETER SELECTION

The training and evaluation process was divided into three main stages. The first stage consisted on defining which base estimators were going to be used for the construction of the gcForest Regressor layers, as well as which models should be used to combine the outputs of the last layer, during the experiments when one such model was used. The second stage in the experiment process involves choosing the best hyper-parameters for each algorithm being trained. The method used on the selection is the same regardless if it is the gcForest Regressor being trained or if it is one of the other models. Stages one and two are described on topics 4.2.1 and 4.2.2, respectively. The third and final stage involves calculating the performance for each trained model.

In the case where a pre-defined architecture is used for training gcForest, the model and parameter selection stages are not necessary. In these cases, only the third stage in the process needs to be executed, as the architecture has already been chosen. This situation arises, for example, when training a gcForest model that has the same configuration as the one proposed in the original article (Zhou and Feng, 2017) or when a single model configuration is used to predict the values for all four time series data.

4.2.1 Model selection

The original gcForest (Zhou and Feng, 2017) consisted of a model constructed by two types of base classifiers, Random Forests and Completely Random Forests. For the model proposed in this study, the equivalent regressors have been implemented. One of the gcForest regressor models was trained using only these two base regressors, in a parallel to the original article - the total number of regressors as well as their parameters were chosen through a randomized search strategy, as described in the next topic. This model will be referred to as “Trees” from this point forward.

Versions of Random Forests models have been proposed for time series forecasting (Qiu et al, 2017), but other machine learning algorithms, such as a

Multilayer Perceptron (MLP) and a Support Vector Regressor (SVR) are more commonly used for this kind of task and have been chosen as benchmark models in a number of studies (De Mattos Neto et al, 2014; De O. Santos Júnior et al, 2019; Tay and Cao 2001). While both models have shown adequate performances in previous studies, the MLP is a neural network - including it in the construction of gcForest would go against the main idea behind the gcForest algorithm , which is to create a deep model without neural nets. For this reason, only the SVR has been implemented as a base regressor. Another version of the gcForest, composed only of SVRs was also trained, and will be called “SVR”.

The authors in (Zhou and Feng, 2017) explain that diversity is one the most important aspects of ensemble models - like gcForest - and as such, tried to incorporate this in the algorithm. Following this line of thought, a third gcForest Regressor model was trained in the context of this study, this time composed by all three of the implemented base regressors. The model composed by these three regressors will be referred to as “All”.

All three of the gcForest Regressor configurations described above can be incremented by adding a final regression model to the last layer, as described in Chapter 3. This last model can be any one of the three previously described base regressors as well as a least squares linear regression (LR). The choice of the linear regression model relates to the original article on stacked regressors (Breiman, 1996), which achieved success by combining the results with such a model.

The parameters for this final model, regardless of what kind of regressor, are also chosen through the randomized search strategy described in the next topic. The exemption to this approach is the Linear Regression, as there are no parameters to be optimized.

4.2.1.1 Model Nomenclature

The gcForest algorithm is composed of two parts: multi-grained scanning and cascade forest. Whenever both stages are applied, the model will be referred to as “gcForest”, however, if only the latter procedure is employed, the model will be referred to only as “Cascade Forest”. This reference will be the first part, out of three, of a model name, as referred to in this study.

This reference will be followed by a mention to which base learners can compose a layer, where “Trees” means that a base learner can be either a Random Forest or a Completely Random Forest, “SVR” refers to to models which the only possible base learner is an SVR and “All” means that base regressors can be any one of the three available: Random Forests, Completely Random Forests or SVR.

The reference to the base learners could be followed by a plus sign (+), which indicates that the results of the last layer of the cascade were combined by a learner instead of using the mean. After the sign there will be an indication of which algorithm was used to combine the results: LR for Linear Regression, RF from Random Forest, Extra for Completely Random Forests or simply SVR for Support Vector Regressor.

As an example, the model “Cascade SVR + LR”, will have only the cascade forest structure, where each layer will be composed only of SVRs, and the results of the final layer will be combined using a linear regression.

4.2.2 Parameter selection

The proposed algorithm, gcForest Regressor, is made up of more than one base model. Deciding which base regressors should be used during the training process is the first step in defining the model architecture. As mentioned previously, the implemented base regressors are the Random Forest Regressor, the Extra Trees Regressor, and the Support Vector Regressor.

Even after selecting which base regressors are to be used, there are still a number of parameters that can be optimized in order to achieve the best possible performance. These parameters include the number of regressors present in each layer, which combination of regressors will be present as well as the hyper-parameter for each regressor model in the architecture. Given the large number of parameters to optimize, a brute force approach such as grid-search would either take a large amount of time or would not cover a sufficient amount of parameters combinations, which could lead to suboptimal model performances.

In order to address these issues and find the best parameter combination, a Randomized Search strategy is used. Through this strategy, a number of combinations are chosen based on a list of parameters to be tuned and the values each of them can assume. This approach has been proved to be equally or even sometimes more effective than a grid-search, while investigating fewer configurations in a larger search space (Bergstra and Bengio, 2013).

The process of defining the best combination can be divided into three phases. The first phase consists in defining the number of base regressors in each layer of the gcForest, while the second phase comprises the definition of which base regressor will form a layer, which can be a Random Forest Regressor, an Extra Trees Regressor, or a Support Vector Regressor depending on the version of the gcForest being trained. These first two steps will determine the basic architecture of the model - its layer setup. This layer architecture will be repeated as many times as the algorithm detects improvement in performance. The third and last phase of this process involves defining the best hyper-parameters in each base regressor. Table 3 describes the values used in the Randomized Search approach for every element in these three phases. When a model is used to combine the final results, a fourth step is present, which consists in defining the best hyper parameters for this final model, which is also done through randomized search. The parameters available during this process are described in Table 4.

The selection of which models will compose a layer is random. This means that it is possible that not every base regressor available to the gcForest will be present in the selection. As such, a model like gcForest All could draw the same configuration as gcForest Trees or gcForest SVR, but with a much smaller probability.

Once a combination of parameters is chosen, a gcForest model is trained on the training set and evaluated on the validation set 30 times. An average mean squared error, obtained through the 30 evaluations is then set as the true performance value for this combination of parameters. This is done in order to account for the fact that some of the models which compose the gcForest are affected by randomness, which can result in different performance values on each iteration.

Like it was previously described, the window size used to model the data is also a parameter to be optimized. The size of the window was not chosen at random as part of the randomized search strategy, but instead tested sequentially, with sizes 2, 5, 10, 15 and 20 being used in a trial.

Table 3 - Descriptions of the parameters used during the randomized search process for the optimization of the gcForest models.

Number of models in each layer	2, 3, 4, 5, 6	
Model	**Parameters**	**Values**
Random Forest Regressor and Extra Trees Regressor	Max Features	‘sqrt’, ‘log2’, ‘auto’
	Optimization Criterion	‘mse’, ‘mae’
	Max Depth	‘None’, 2, 5, 10, 15
	Min. Samples in Leaf	1, 2, 5, 10
	Number of Trees	5, 10, 25, 50, 100
SVR	Gamma	1, 0.1, 0.01, 0.001, 0.0001
	C	0.01, 0.1, 1, 10, 100, 1000, 10000

Source: Author, 2020.

Table 4 - Descriptions of the parameters used during the randomized search process for the optimization of the final model.

Model	**Parameters**	**Values**
Random Forest Regressor and Extra Trees Regressor	Max Features	‘sqrt’, ‘log2’, ‘auto’
	Optimization Criterion	‘mse’, ‘mae’
	Max Depth	‘None’, 2, 5, 10, 15
	Min. Samples in Leaf	1, 2, 5, 10
	Number of Trees	5, 10, 25, 50, 100, 250, 500
SVR	Gamma	1, 0.1, 0.01, 0.001, 0.0001
	C	0.01, 0.1, 1, 10, 100, 1000, 10000

Source: Author, 2020.

As a result of this setup, a total of 9600 models were trained for each version of gcForest in this study - 64 parameter combinations, trained 30 times each, for 5 different window sizes. The exception was the SVR - given the deterministic nature of the model (training with the same data and parameters will yield identical results), the SVR was only trained one time for each combination, instead of 30, totaling 320 runs.

The randomized search strategy employed in the experiments reduces the parameter search space by a significant margin. Table 5 shows the comparison between the original search space when considering the chosen parameters and the search space that had to be covered if a Grid Search Strategy had to be chosen. The only model for which a reduction did not occur was the single SVR, because the total possible configurations amounted to 35, which is less than the 64 possible choices. Reductions are larger for the proposed models when compared to the single algorithms. The single models and their parameters will be further explained in Section 4.2.4. This table does not include the case when an intelligent model is used for combining the inputs in the last layer. In this scenario, the search space reduction would be larger, as the original search space increases while the chosen one does not.

Table 5 - Original and Chosen Search Space for parameter tuning.

Model	Original Search Space Size	Chosen Search Space Size	Search Space Reduction
Random Forest	600	64	89.3%
MLP	108	64	40.74%
SVR	35	64	-
Cascade Trees	6000	64	98.93%
Cascade SVR	175	64	63.43%
Cascade All	6175	64	98.96%

Source: Author, 2020. The original space comprises all possible parameter combinations, and the chosen space is the actual number of configurations investigated.

4.2.3 Measurements of Performance

After the selection of the best parameters, the training and validation dataset were combined and another 30 versions of the best model were trained using this combination. Every run was evaluated on the test set according to three performance metrics, described in section 4.3. The average value for each metric represents the final model performance. Just like in the optimization phase, different training runs for the same model were conducted in order to account for the randomness present in each base regressor, with the exception of the SVR.

When a predefined architecture was used, the procedure to obtain the final model performance was the same - the final result was the average of 30 model performances, with these models being trained on the training and validation set combined.

4.2.4 Machine Learning Models for Comparison

Aiming to compare the performance of the proposed method to other machine learning algorithms, three single models were trained on the same datasets and their training followed the same procedures as those of gcForest.

As the basis for the original gcForest model, Random Forests were trained as single models. Because the SVR was added as a possible base regressor for gcForest and since it is commonly used in time series forecasting, including as a benchmark model, it was also included in the list of algorithms that gcForest will be compared to.

Seeing that an MLP is a neural network and gcForest was proposed as an alternative to deep neural networks, it was not adequate to include a MLP as a base regressor. On the other hand, this is a model which has been used extensively for time series forecasting and will thus be added as a comparison source.

All single models which will be used as comparison were trained with the same randomized search strategy as gcForest and described in previous sections. After the best combination of parameters were chosen, the models were retrained by combining the training and validation datasets and the final performance was obtained by averaging 30 runs. Table 6 describes the parameters available during the randomized search for each of the single models. In cases where the model is also a base regressor for gcForest, the parameters are the same.

Table 6 - Descriptions of the parameters used during the randomized search process for the optimization of single models.

Model	Parameter	Values
Random Forest Regressor	Max Features	‘sqrt’, ‘log2’, ‘auto’
	Optimization Criterion	‘mse’, ‘mae’
	Max Depth	‘None’, 2, 5, 10, 15
	Min. Samples in Leaf	1, 2, 5, 10
	Number of Trees	5, 10, 25, 50, 100
MLP Regressor	Initial Learning Rate	0.00001, 0.0001, 0.001
	Activation Function	‘logistic’, ‘tanh’, ‘relu’
	Nodes in Layer	1, 5, 10, 15, 20, 50
	Solver	‘Lbfgs’, ‘adam’
	Max Iterations	100000
SVR	Gamma	1, 0.1, 0.01, 0.001, 0.0001
	C	0.01, 0.1, 1, 10, 100, 1000, 10000

Source: Author, 2020.

4.3 EVALUATION METRICS

All models investigated during this study were evaluated according to three different metrics. These metrics were chosen because each one of them was used as a performance metric in previous time series forecasting studies. For all of them, a lower value indicates better performance.

The first metric used to evaluate the models is the Mean Squared Error (MSE) - the equation for this metric was presented in Chapter 3, section 3.3.1.

The second metric chosen to evaluate the performance of the models in this study is the Mean Absolute Percentage Error (MAPE), obtained through the equation

2, available below, and also used in previous studies for time series forecasting, including one study where a version of a gcForest Regressor was proposed (Yin et al, 2020). For the case when *t(i)* is zero, the value zero is considered in the sum.

$$MAPE = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{t(i) - p(i)}{t(i)} \right| \tag{2}$$

The third and last performance metric is the Mean Absolute Error (MAE). The equation to this metric is similar to the one for MSE, with the distinction being that for MAE, the difference between the predicted value and the true value is not squared, which means that the error can most directly relate to the values in the time series. On the other hand, for MSE, larger errors have a higher impact on the final results, since the difference between predicted and true is squared. The metric MAE has also been used before in order to evaluate time series forecasting model performance (Cao and Tay, 2003; De O. Santos Júnior et al, 2019) and it's equation is shown in equation 3.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |t(i) - p(i)| \tag{3}$$

For all equations presented here, *t(i)* is the true value of the *i-th* element, while *p(i)* is the predicted value for the same element, within a dataset of *n* elements in total.

4.4 CODE IMPLEMENTATION AND AVAILABLE HARDWARE

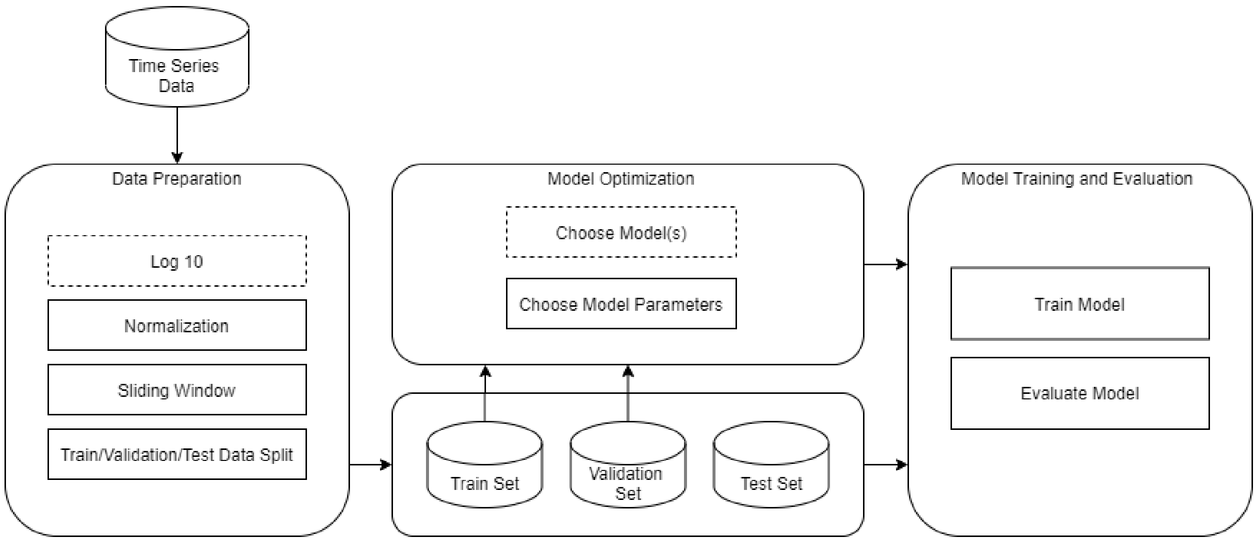
The code for all the experiments was written in Python 3.8. In order to alter gcForest for regression, the code provided by (Zhou and Feng, 2017) was used as a base. All other models were implemented using the scikit-learn open-source library (Pedregosa et al, 2011). Whenever a parameter value is not mentioned, default values for this library should be assumed.

Experiments were run on a notebook running Windows made up of an Intel Core i7 Processor and 16Gb of RAM. The notebook contains a NVIDIA GeForce MX250 2Gb Graphics Card, but experiments could not be run on GPU because the library that implements the models, scikit-learn, does not offer GPU support.

4.5 CHAPTER OVERVIEW

This chapter described the process involved in the training and evaluation of the gcForest model. The complete experiment process can be summarized in a sequence of three stages: Data Preparation, Model Optimization and Model Training and Evaluation. Figure 9 shows an overview of the complete process. The first stage, or the Data Preparation stage, consists of transforming the data, modelling it as a supervised machine learning regression problem and splitting it into training, validation and testing sets.

Figure 9 - Overview of the experiment process.



Source: Author, 2020. Dotted boxes represent steps which are note mandatory and can apply depending on the dataset or the model being trained.

The second stage will define the basic structure of gcForest. This can either mean that a fixed model configuration will be established or that the best hyper-parameters for the model being trained will be chosen using a randomized search approach. At this stage, the base regressors that will compose the gcForest model will be selected as well as the parameters that define these regressors. The same randomized search approach was applied to the process of training the single models used in comparison to the proposed gcForest, with the exception that the model selection step was not necessary.

After selecting the best model configuration, the algorithm is trained and evaluated on three metrics: MSE, introduced in the last previous chapter and further discussed here, Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE). Each step of the process will be explained in detail in the next sections.

5 EXPERIMENTS RESULTS

This chapter presents the results obtained from the execution of the experiments described in the previous chapter. Section 5.1 shows a comparison of the original gcForest configuration, both with and without the multi-grained scanning to traditional machine learning algorithms. The following sections, 5.2 to 5.5, describe the performance of the proposed algorithm for each of the four analyzed time series, as well as a comparison to other traditional models. Results are shown for the normalized data points of the set.

5.1 PERFORMANCE OF THE ORIGINAL CONFIGURATIONS

One of the main advantages of gcForest was that the same configuration of parameters was able to achieve better performance when compared to other traditional models (Zhou and Feng, 2017). In order to test this approach, all four series were used to train gcForest regressors which had the original model architecture: a random forest and a completely random forest with 500 trees each for the multi-grained scanning stage, as well as four random forests and four completely random forests also with 500 trees each for the cascade structure.

For this experiment, a sliding window of size 20 was used for all the four series. The second type of sliding window procedure, which is part of the multi-grained scanning algorithm, had sizes *d*/16, *d*/8 and *d*/4, where *d* is the number of original features, just like in the original model.

Tables 7-9, show the results for both the complete procedure, as well as that of only the cascade forest stage, as one of the hypotheses in this study was that, for the cas of time series forecasting, the steps of the multi-grained scanning stage might not improve model performance. On Table 7 results for MSE are presented, while Tables 8 and 9 show the results for MAE and MAPE, respectively. For all three tables, the values shown in brackets are the standard deviations for the models 30 iterations - models like the SVR do not depend on random decision and thus do not vary.

Table 7 - MSE results for the original configurations of gcForest on the normalized test set.

Model	Dataset			
	Lynx	S&P500	Star Brightness	Sunspot
gcForest	4.01E-2 (1.02E-3)	3.78E-2 (6.62E-6)	1.46E-2 (1.1E-4)	4.83E-2 (3.96E-4)
Cascade Forest	2.52E-2 (6.71E-4)	3.79E-2 (8.45E-6)	5.9E-4 (2.14E-5)	1.7E-2 (2.27E-4)
Random Forest	9.99E-3 (4.75E-4)	3.85E-2 (4.24E-4)	3.42E-4 (9.36E-6)	1.91E-2 (3.19E-4)
MLP	1.43E-2 (8.69E-3)	**2.25E-4 (1.56E-4)**	**1.73E-4 (1.41E-5)**	**1.18E-2 (8.62E-4)**
SVR	**7.39E-3 (-)**	1.16E-2 (-)	1.81E-3 (-)	1.48E-2 (-)

Source: Author, 2020. Values shown in brackets are the standard deviations for the 30 iterations of the model.

Table 8 - MAE results for the original configurations of gcForest on the normalized test set.

Model	Dataset			
	Lynx	S&P500	Star Brightness	Sunspot
gcForest	1.81E-1 (2.14E-3)	1.59E-1 (2.03E-5)	9.59E-2 (3.36E-4)	1.64E-1 (6.52E-4)
Cascade Forest	1.20E-1 (2.56E-3)	1.59E-1 (2.60E-5)	1.81E-2 (2.30E-4)	9.17E-2 (4.50E-4)
Random Forest	7.66E-2 (2.06E-3)	1.61E-1 (1.30E-3)	1.50E-2 (2.62E-4)	9.53E-2 (9.71E-4)
MLP	9.90E-3 (2.15E-2)	**1.15E-2 (3.53E-3)**	**1.10E-2 (3.41E-4)**	**8.16E-2 (3.77E-3)**
SVR	**6.35E-2 (-)**	1.05E-1 (-)	3.47E-2 (-)	8.77E-2 (-)

Source: Author, 2020. Values shown in brackets are the standard deviations for the 30 iterations of the model.

Table 9 - MAPE results for the original configurations of gcForest on the normalized test set.

Model	Dataset			
	Lynx	S&P500	Star Brightness	Sunspot
gcForest	41.23 (0.79)	17.68 (0.01)	27.84 (0.13)	80.36 (0.48)
Cascade Forest	23.70 (1.05)	17.72 (0.01)	5.61 (0.07)	34.98 (0.23)
Random Forest	17.81 (0.59)	17.94 (0.16)	5.14 (0.10)	17.81 (0.59)
MLP	21.93 (6.14)	**1.35 (0.39)**	**3.78 (0.26)**	21.93 (6.14)
SVR	**15.92 (-)**	12.57	10.62 (-)	**15.92 (-)**

Source: Author, 2020. Values shown in brackets are the standard deviations for the 30 iterations of the model.

As is shown on the tables, regardless of the chosen metric, the Cascade Forest structure alone displays better performance when compared to that of the complete gcForest model. The only dataset in which the addition of multi-grained scanning appears to be useful in S&P 500, but the gained performance might not warrant the increased training and processing time for the model.

Nevertheless, even with a superior performance, the original configuration of cascade forest is not enough to beat traditional models optimized by the randomized search procedure described in Chapter 4, specially the MLP and SVR, which had the best performances in three (S&P 500, Star Brightness and Sunspot) and one series (Canadian Lynx), respectively.

Taking this results into account, further experiments were conducted for all four time series, this time introducing the randomized search procedure for cascade forest, which appear to perform better than the complete procedure, as well as the other models, while also modifying the aggregation method for the models in the last layer - by changing it from and average to a combination by learner. The following sections in the chapter will delve deeper into the results considering these conditions.

5.2 CANADIAN LYNX

The results for the prediction on the Canadian Lynx time series is presented in Table 10. Along with the traditional machine learning models, all three configurations of the cascade forest discussed in Chapter 4 are also present. Also present in Table 10 are the combinations by machine learning model which yielded the best results. For the sake of clarity and in order to guarantee a better visualization of results, only the models with the best performances are shown for each type of cascade combination. The results from all combinations, including the results shown in Tables 7-9, can be found in Appendix A. The parameter configuration for the models with the best performance can be found on Appendix B.

Considering only the variations of cascade forest which average the results of the last layer of the cascade structure, none of the proposed models had the best performance, with the traditional SVR showing the best results. On the other hand, when the combination strategy is a Linear Regression model, the Cascade SVR + LR algorithm performs better than all the other models for all three metrics.

When taking the standard deviations into account, there are overlaps of the Cascade SVR + LR for all three metrics: with the MLP for MSE and with the Cascade All + LR for MAE.

Figure 10 illustrates the scenario for the mean absolute error. Different from the case with the MSE, the two Cascade configurations do seem to overlap, making it impossible to infer if the results are indeed different through this method.

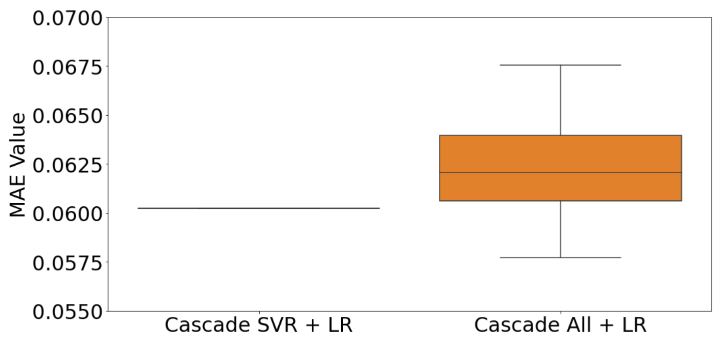
Figure 11 shows the boxplots for the situation described by the MSE. The actual boxplots for the two models, MLP and Cascade SVR + LR do not overlap in any way, which indicate the samples are indeed drawn from different distributions, with the second model averaging a lower MSE. A higher mean and standard deviation were a result of a single outlier for the MLP model, with a MSE of 6.11E-2. The same outlier was also observed in the calculation of the MAE and MAPE metric, but it did not affect the mean metric values to the same degree.

Table 10 - Prediction performance for the Canadian Lynx time series.

Model	MSE	MAE	MAPE
Random Forest	9.99E-3 (4.75E-4)	7.66E-2 (2.06E-3)	17.81 (0.59)
MLP	1.43E-2 (8.69E-3)	9.90E-2 (2.15E-2)	21.93 (6.14)
SVR	7.39E-3 (-)	6.35E-2 (-)	15.92 (-)
Cascade Trees	1.12E-2 (1.25E-3)	7.92E-2 (5.03E-3)	18.17 (1.51)
Cascade SVR	7.37E-3 (-)	6.35E-2 (-)	15.46 (-)
Cascade All	1.08E-2 (9.76E-4)	7.59E-2 (4.73E-3)	17.98 (1.28)
Cascade Trees + LR	9.88E-3 (5.90E-4)	7.30E-2 (3.10E-3)	16.08 (0.94)
Cascade SVR + LR	**7.02E-3 (-)**	**6.02E-2 (-)**	**14.59 (-)**
Cascade All + LR	7.50E-3 (3.08E-4)	6.23E-2 (2.35E-3)	15.53 (0.80)

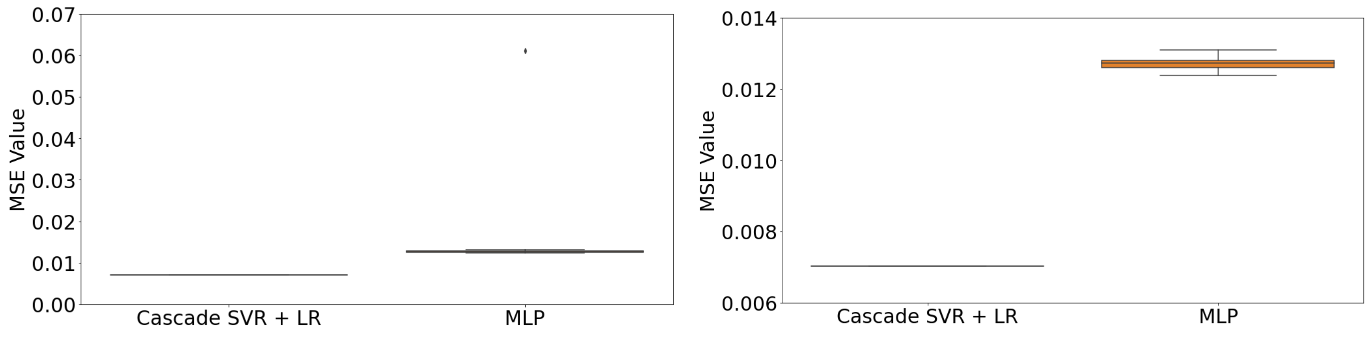
Source: Author, 2020. Results represent the performance on the normalized test set and the values shown in brackets are the standard deviations for the 30 iterations of the model.

Figure 10 - Boxplots for the MAE metric of best models for the Canadian Lynx time series.



Source: Author, 2020.

Figure 11 - Boxplots for the MSE metric of best models for the Canadian Lynx time series.

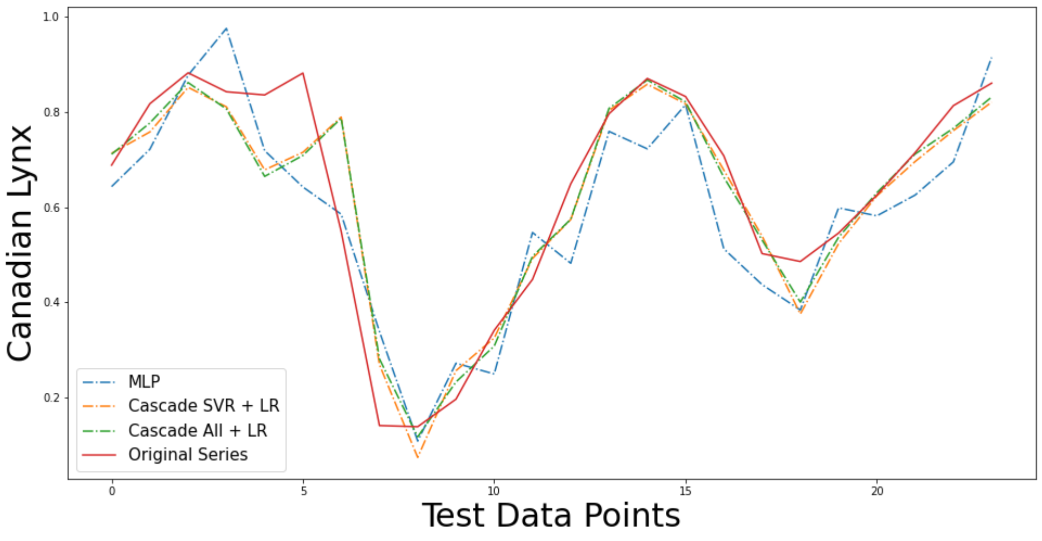


Source: Author, 2020. Both models are shown in full view, as well as a close-up without the offending outlier.

From the results it is possible to see that the Cascade SVR + LR model performed better than all the others in at least two metrics, MSE and MAPE. For MAE, it was not possible to determine if the best alternative was indeed the Cascade SVR + LR or the Cascade All + LR. Overall, models with the proposed cascade structure performed better than all the other analyzed algorithms.

Figure 12 shows the plot of the original test set for Canadian Lynx, as well as the predicted values from one iteration of the model with the best mean results (Cascade SVR + LR) and for those which approximate these results when considering the standard deviation. From the image, it is possible to notice that the prediction for the MLP (in blue), really appears to be worse than that of the cascade models, like the blox plot suggested. On the other hand, both cascade models indicated by MAPE as the best performing ones more closely follow the original data, and each other, making it impossible to decide which one is best from the plot alone.

Figure 12 - Plot of the predicted series for the Canadian Lynx dataset.



Source: Author, 2020.

5.3 STAR BRIGHTNESS

Table 11 is a representation of the results for the forecasting on the test set of the Star time series. The Cascade SRV + LR again shows the best results among the investigated models. For this particular time series, the performance of the best model does not fall within the interval of any other algorithm results when the standard deviation is taken into account. The results from all investigated models can be found on Appendix A, where the parameter configuration for the best model can be found on Appendix B.

The best configuration for the Cascade Trees ensemble is the one where results from the final layer were combined by a completely random tree model. This is the first time during the results analysis where such a model is chosen as the combination learner. Before, all the best results were obtained by combining with a linear regression.

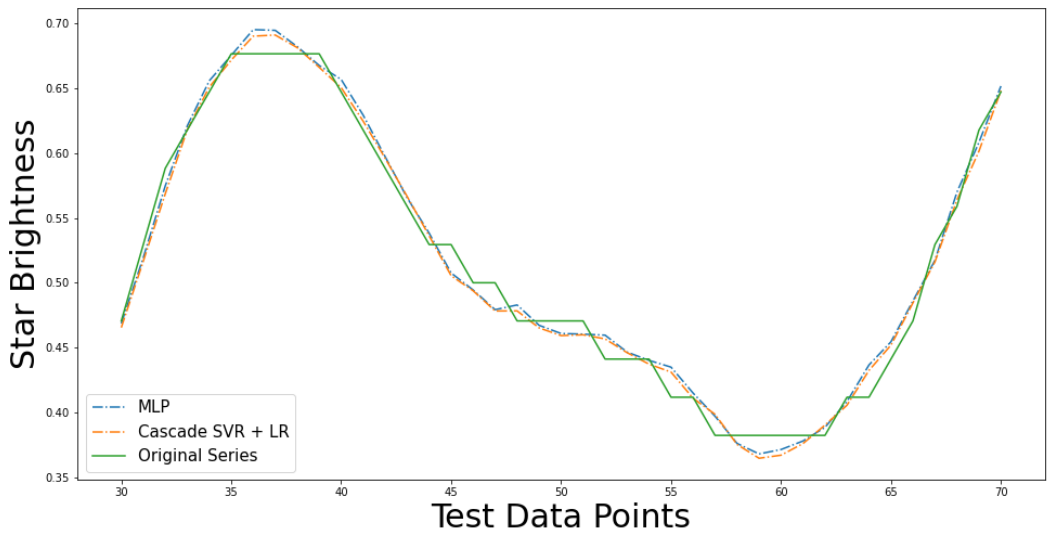
Figure 13 represents the plot for the predicted values for the Star Brightness test set for the two best performing models: Cascade SVR + LR, which had the best results for all metrics, and the MLP, which is the model that came in second in all metrics. The MLP plot represents the results from one iteration of the model. The plot is a close-up encompassing points 30 to 70, out of the 145 used for testing. The close-up was necessary in order to allow for the observation of the performance for both models, as they very closely followed the original series and each other.

Table 11 - Prediction performance for the Star Brightness time series on the normalized test set.

Model	MSE	MAE	MAPE
Random Forest	3.42E-4 (9.36E-6)	1.50E-2 (2.62E-4)	5.14 (0.10)
MLP	1.73E-4 (1.41E-5)	1.10E-2 (3.41E-4)	3.78 (0.26)
SVR	1.81E-3 (-)	3.47E-2 (-)	10.62 (-)
Cascade Trees	3.35E-4 (1.08E-5)	1.41E-2 (3.13E-4)	4.78 (0.14)
Cascade SVR	1.96E-3 (-)	3.71E-2 (-)	15.47 (-)
Cascade All	4.17E-4 (1.79E-5)	1.52E-2 (2.92E-4)	4.89 (0.20)
Cascade Trees + Extra	3.32E-4 (9.47E-6)	1.39E-2 (2.70E-4)	4.68 (0.13)
Cascade SVR + LR	**1.42E-4 (-)**	**9.88E-3 (-)**	**3.35 (-)**
Cascade All + LR	2.51E-4 (1.13E-5)	1.28E-2 (2.71E-4)	4.27 (0.12)

Source: Author, 2020. Values shown in brackets are the standard deviations for the 30 iterations of the model.

Figure 13 - Plot of the predicted values for the Star Brightness dataset.



Source: Author, 2020.

5.4 SUNSPOT

Table 12 displays the results obtained from the forecasting models for the test set of the sunspot time series.

For this time series, the Cascade SVR model with the original combination method - the mean - was the one to achieve the best performance in two out of three metrics, MSE and MAE. The values for these metrics do not fall into the interval of other results when the standard deviation is taken into account, which implies that the Cascade SVR might actually be the superior model regardless of the effect of randomness. This was the only instance in which combining the final layer results by means of a learner did not yield a better performance than using the mean. It is also the first time where the SVR was chosen as the best combination for a model (for the Cascade SVR+SVR), even if using it did not improve the performance of the original.

When it comes to MAPE, the model with best performance was the Cascade All where the final results are combined by the mean. According to the margin established by the standard deviation values, the model's performance could be challenged by other configurations: Cascade All, combined by random forest and Cascade Trees, combined by completely random forest.

Table 12 - Prediction performance for the Sunspot time series on the normalized test set.

Modell	**MSE**	**MAE**	**MAPE**
Random Forest	1.91E-2 (3.19E-4)	9.53E-2 (9.71E-4)	33.59 (0.61)
MLP	1.18E-2 (8.63E-4)	8.16E-2 (3.77E-3)	34.37 (2.29)
SVR	1.48E-2 (-)	8.77E-2 (-)	43.60 (-)
Cascade Trees	1.92E-2 (9.91E-4)	9.61E-2 (2.30E-3)	34.51 (1.24)
Cascade SVR	**9.56E-3 (-)**	**7.60E-2 (-)**	42.29 (-)
Cascade All	1.50E-2 (6.67E-4)	8.30E-2 (1.86E-3)	**29.49 (0.75)**
Cascade SVR + SVR	1.18E-2 (-)	7.99E-2 (-)	40.53 (-)
Cascade Trees + Extra	1.61E-2 (6.01E-4)	8.57E-2 (1.75E-3)	30.05 (0.95)
Cascade All + RF	1.63E-2 (7.80E-4)	8.77E-2 (2.36E-3)	31.92 (1.11)

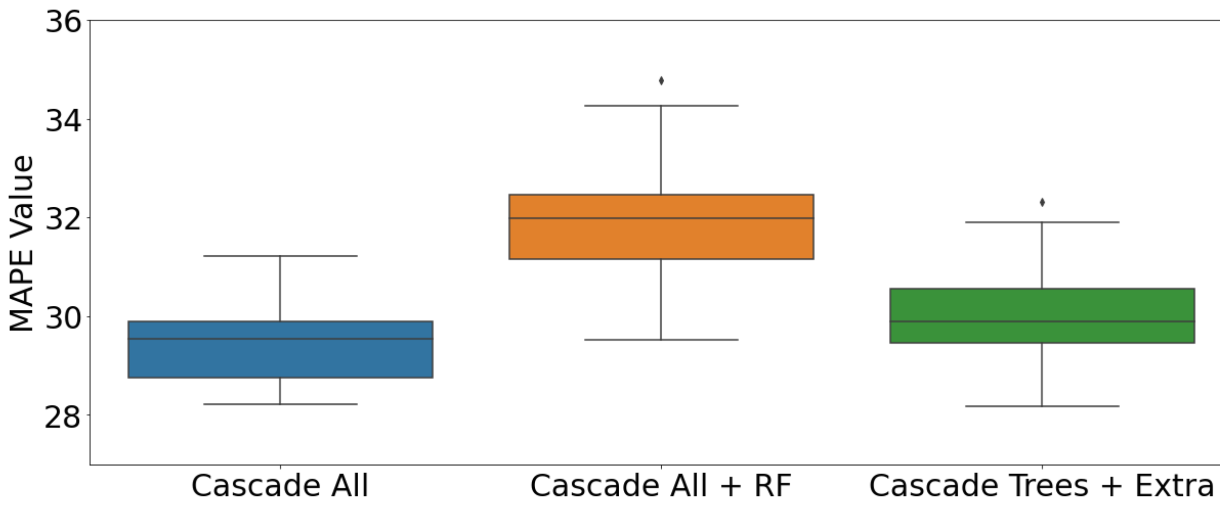
Source: Author, 2020. Values shown in brackets are the standard deviations for the 30 iterations of the model.

Figure 14 shows the boxplot for all three configurations on MAPE in order to compare the distributions. From the plot, it is possible to see that values for all models are not completely different, with the edge of the lower quartile Cascade All + RF reaching the edge of the upper quartile for the Cascade Trees + Extra and the boxes for Cascade All and Cascade Trees + Extra overlapping. This situation makes it difficult to tell which model has better performance or if they are in fact equivalent and a sample size larger than 30 would reveal this equity.

Even without being able to tell which model is the best according to the MAPE metric, the Cascade SVR algorithm performed better on the other two metrics, which makes it possible to state that this might be the best performing model for this task.

It is notable that the winning configuration combines the results of the last layer using the mean. No other Cascade SVR combinations were found through the randomized search strategy that could outperform this type of combination. All configurations and their representative results can be found on Appendix A. The parameter configuration for the models with the best performance can be found on Appendix B.

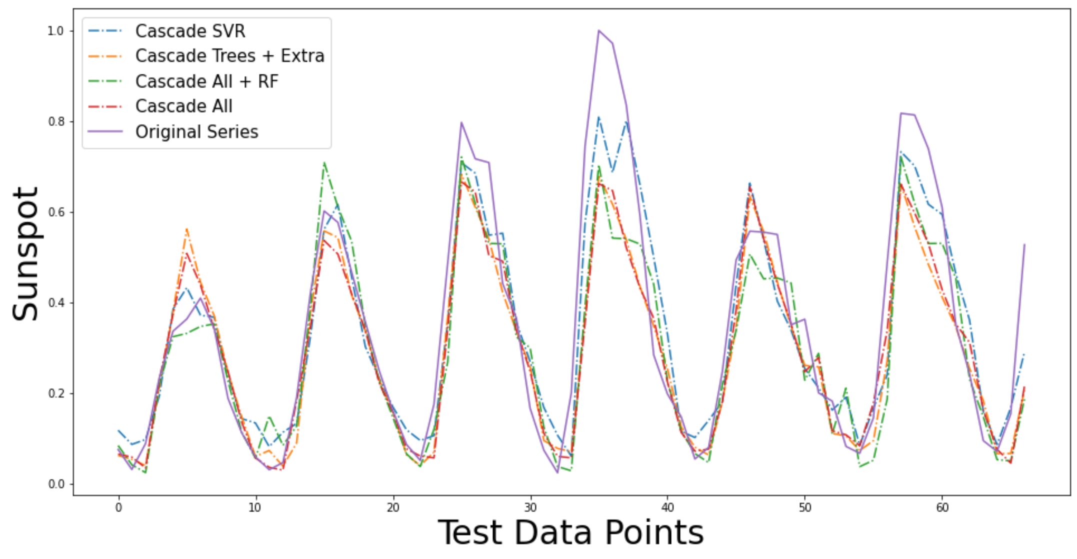
Figure 14 - Boxplot of MAPE values for the three models whose performances seem to overlap for Sunspot.



Source: Author. 2020.

Figure 15 depicts the predictions of one iteration of the Cascade SVR model, which had better performance according to MSE and MAE, as well as one of Cascade All + RF, Cascade Trees + Extra and Cascade All, which had lower values for MAPE. Cascade SVR, depicted in a blue dashed line, appears to better predict local maximums, which might have led to the overall better performances in two out of three metrics.

Figure 15 - Model predictions for the Sunspot time series test set.



Source: Author, 2020

5.5 S&P 500

The results for the investigated models on the test set for the S&P 500 time series data are presented on Table 13. Once again the Cascade SVR + LR model achieves the best performance out of all 9 analyzed algorithms, for all of the three established performance metrics. The performance for all investigated models can be found in Appendix A, while the parameter configuration for the best performing models can be found on Appendix B.

Table 13 - Prediction performance for the S&P 500 time series.

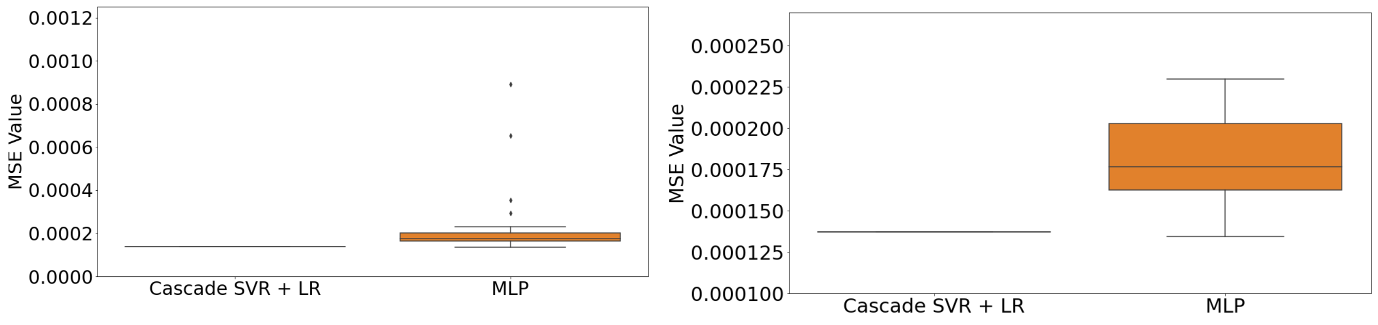
Model	MSE	MAE	MAPE
Random Forest	3.85E-2 (4.24E-4)	1.61E-1 (1.30E-3)	17.94 (0.16)
MLP	2.25E-4 (1.56E-4)	1.15E-2 (3.53E-3)	1.35 (0.39)
SVR	1.16E-2 (-)	1.05E-1 (-)	12.57 (-)
Cascade Trees	3.80E-2 (9.11E-5)	1.60E-1 (2.83E-4)	17.76 (0.03)
Cascade SVR	1.51E-2 (-)	1.20E-1 (-)	14.34 (-)
Cascade All	2.18E-2 (5.99E-5)	1.32E-1 (2.24E-4)	15.14 (0.03)
Cascade Trees + LR	3.76E-2 (1.04E-4)	1.59E-1 (3.20E-4)	17.62 (0.04)
Cascade SVR + LR	**1.37E-4 (-)**	**9.82E-3 (-)**	**1.17 (-)**
Cascade All + LR	1.51E-3 (-)	3.06E-2 (-)	3.40 (-)

Source: Author, 2020. Results represent the performance on the normalized test set and the values shown in brackets are the standard deviations for the 30 iterations of the model.

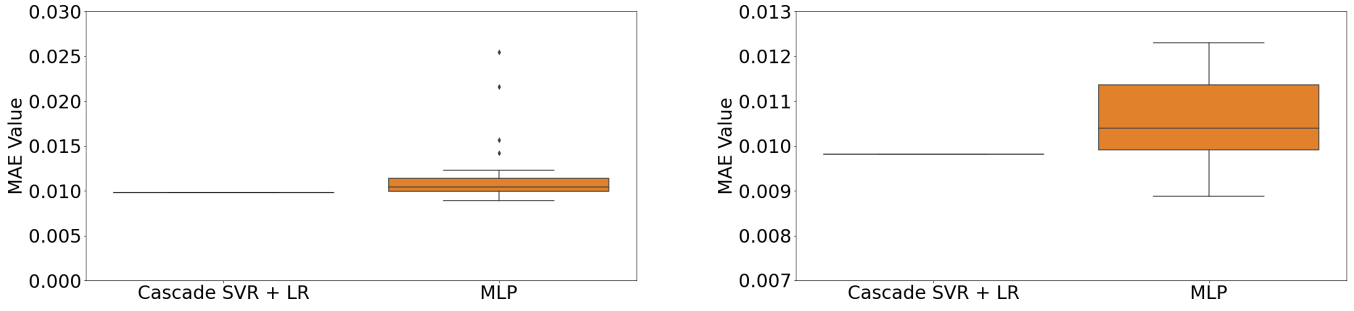
The results for the MLP MSE, MAE and MAPE, when considering the standard deviation values, encompass the results of the Cascade SVR + LR. Figure 16 shows the boxplot for the 30 iterations of each metric. In the case of the MSE, the samples appear to actually be different as the point of intersection between the boxplots occurs only by the edge of the lower whisker. On the other hand, such a difference is not so clear for the MAE and MAPE plots. In both cases, the plot intersection occurs not on the edge of the lower whisker, but instead on the edge of the lower quartile, not showing clear evidence of a difference in performance. A case could be made for MAE, as the MLP median is well above the value for Cascade SVR + LR, which could be a lead in indicating that the latter model showed better performance. The same can not be said for MAPE, as both medians appear to be quite close.

An interesting result can be observed for the Cascade All + LR model, for which no standard deviation value is found. This happened because, according to the randomized search process, the best combination of algorithms included only SVRs.

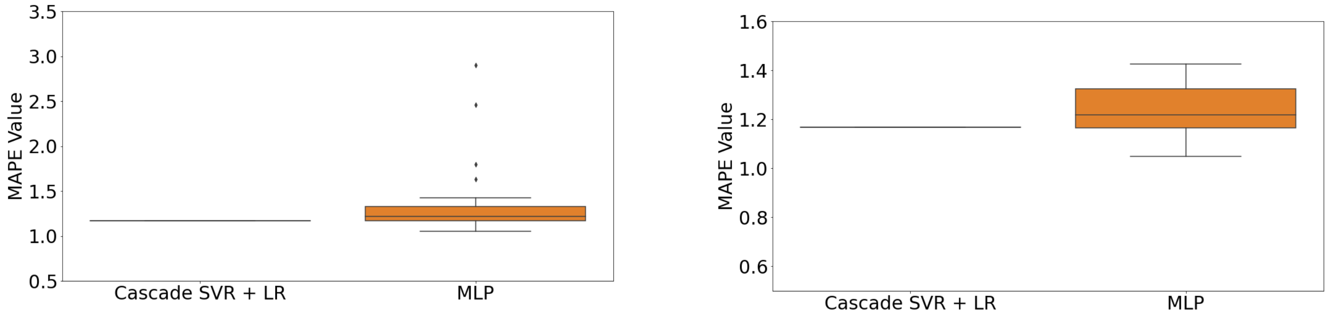
Figure 16 - Boxplots for the distribution of all three metrics for the S&P 500 dataset, regarding the two models with better performance.



(a)



(b)



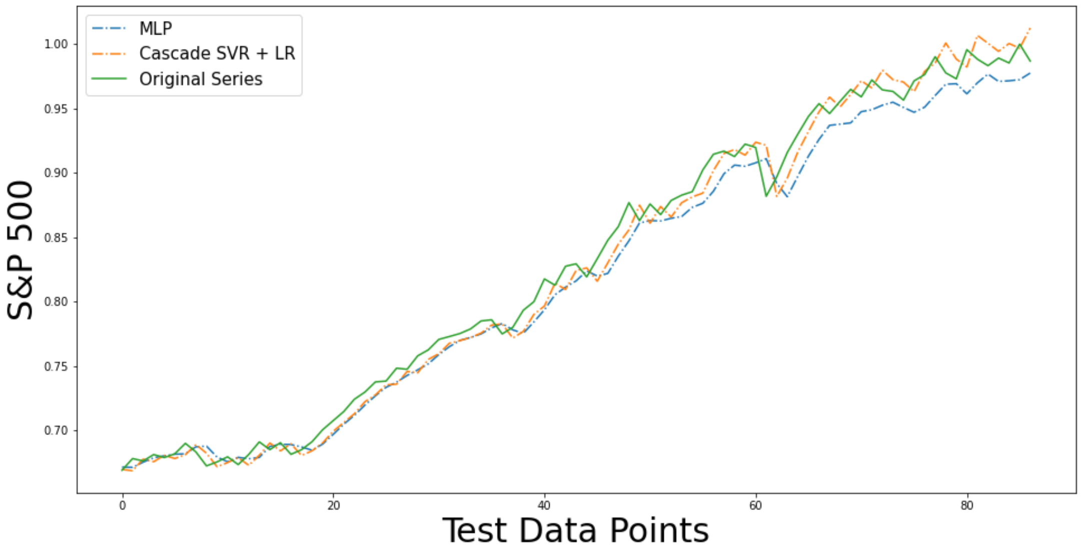
(c)

Source: Author, 2020. In (a), the complete boxplot for MSE including outliers, as well as a closeup excluding them. In (b) and (c) the same information is displayed for MAE and MAPE, respectively.

Given the discordance between metrics results, it is not possible to state which model has a definite best performance: MLP or Cascade SVR + LR, even if the latter show a lower mean for all metrics. Figure 17 shows the plot for the predicted values in the test set for one iteration of the best performing models - Cascade SVR +LR and MLP. The performance of the latter model appears to decrease for the observations at the end of the series. While this behaviour is not as explicitly clear for the Cascade SVR + LR model, predicted values in this model appear to move away from the original values for these observations as well.

This behaviour can be better explained when the complete series is taken into account. Figure 8, presented in the last chapter, shows the plot of the complete S&P500 time series, highlighting the observations used for the test set. The observations that make up the last points have higher values than any of the previously seen data, which might explain the inability of these models to correctly predict the values in this area.

Figure 17 - Plot of predicted values for S&P 500 time series dataset.



Source: Author, 2020.

5.6 DISCUSSION

The traditional gcForest algorithm, which is composed of two stages, multi-grained scanning and cascade forest showed good performance when compared to other models for a range of classification problems by using the same configuration of parameters. This configuration showed inferior results for the time series forecasting tasks in comparison to an MLP, SVR and Random Forest.

Also, the full model, which contained the multi-grained scanning procedure, was inferior to the cascade forest for almost all time series, with the exclusion of the S&P 500. For this reason, further experiments were conducted using only the Cascade Forest algorithm. The inadequacy of the multi-grained scanning procedure was one of the hypotheses raised in this study. Results confirm that this step in the algorithm, as it was proposed for classification, is not a good fit for regression, specifically problems involving time series forecasting.

The four time series were analyzed individually and the results for the traditional models as well as those of the best configurations of proposed models.

In three out of four time series discussed, a version of the proposed model exhibited a better performance when compared to the other models. For all three datasets, the best performing model involved the Cascade SVR: while the Cascade SVR + LR for the Lynx and Star Brightness datasets and the Cascaded SVR, combined by mean, for the Sunspot time series. This is in accordance with evidence found in literature, where an SVR is present as a time series forecasting model with much more frequency than tree based algorithms. In fact, combinations containing only SVRs were chosen as the best mode, through the randomized search strategy, even when other models were available, as was the case for S&P 500 dataset.

Regarding the S&P 500 time series dataset, the best model according to the mean of all three metrics, also appears to be the Cascade SVR + LR. This cannot be confirmed because results for all metrics overlap when the standard deviation from the models 30 iterations are taken into account. Boxplot images can confirm the better performance for Cascade SVR + LR in regards to MSE, but are unclear for the other two metrics, MAE and MAPE.

In regards to the mode of combination for the model in the last layer of the cascade forest, a simple Linear Regression appears to be the best choice in most cases, with some expectations where the Cascade Trees structure was combined with a completely random tree model.

5.7 CHAPTER OVERVIEW

This chapter presented the results originating from the experiments described in Chapter 4. Results validate the hypothesis that the multi-grained stage, such as it was proposed for classification, does not improve the performance of Cascade Forest. Further experiments conducted only with Cascade Forest showed that at least one configuration of the proposed model exhibited better performance when compared to traditional models, for three of the four time series: Canadian Lynx, Star Brightness and Sunspot. While the best mean value for all three metrics in the S&P 500 series belongs to one of the proposed models, there is not enough evidence to guarantee that this model performs better than an MLP, considering the latter model's variations due to randomness.

6 CONCLUSIONS AND FUTURE WORK

This chapter talks about the work presented in this study as well as indicates possible directions for further investigations involving the proposed model.

6.1 CONCLUSIONS

Time series forecasting is an area of study with applications in numerous fields, such as finance, medicine and commerce. Forecasting methods range from models such as ARIMA, capable of modelling linear behavior in the data, to algorithms involving machine learning, for which modelling of non-linear behavior is possible.

This dissertation proposes a machine learning approach to time series forecasting which adapts gcForest (Zhou and Feng, 2017), an algorithm first proposed to solve classification problems, for regression. Apart from the original architecture of gcForest, also studied is the possibility of adding an additional model as a way of combining the final results of gcForest.

One of the hypotheses of this study is that the multi-grained scanning procedure, as it was proposed for classification, does not improve performance when applied directly to regression, specially time series forecasting. This hypothesis proved true when the performance of the model with and without the procedure were compared and the cascade forest structure alone achieved better results.

Further experiments conducted with Cascade Forest and involving a randomized search strategy in order to optimize the model's parameters showed promising results. Performance of the optimized Cascade Forest was better than that of the models used for comparison for three out of the four series investigated. The fourth time series dataset, S&P 500, was the only one in which Cascade Forest did not achieve a significant performance improvement. S&P 500 is a financial time series, which are knowingly challenging to forecast (Kim, 2003).

One of the most important advantages of gcForest is the fact that a single configuration showed good performance for a number of classification tasks, when compared to other models. However, the original configuration of parameters for gcForest, even when considering only the Cascade Forest structure, did not perform better than traditional models for time series forecasting, but this does not mean that no such configuration exists. The Cascade SVR + LR model displayed good performance for most of the time series investigated, which could be an indication of the desired components for such configuration.

As the results show, the performance of gcForest, especially Cascade Forest, when applied to time series can be better than that of other models. However, in some applications, training and prediction times should be taken into consideration. Because an instance has to go through all levels of the cascade and multi-grained scanning can generate thousands of new instances, time and memory uses can be large for gcForest (Pang et al, 2018). The proposed method could be employed in cases where prediction accuracy is more important than training and testing times.

6.2 FUTURE WORK

As mentioned by (Zhou and Feng, 2017), research with deep models that do not involve neural networks is just starting. This is especially true when this type of algorithms are applied to regression and even more so to time series forecasting.

One of the most important improvements that could be made to the proposed model is the introduction of an alternative to the multi-grained scanning procedure. The strategy employed for classification does not seem to work well on regression but, as a step designed to benefit datasets where features are related, this procedure would be beneficial to problems involving time series.

Another form of obtaining better performance would be through the tuning of model parameters, through a strategy other than randomized search, such as genetic algorithms. More intelligent approaches to choosing parameter values as well the models which will compose the architecture of gcForest could be a great benefit to the model, as results indicate that the type of base learner greatly impacts performance.

Finally, in regards to the base learners that compose gcForest, it is possible that an increase in performance could also be achieved by including linear models into the architecture of a cascade layer. The combination of linear and non-linear models in time series forecasting might increase model performance. One possibility is to add ARIMA predictions as a feature between cascade layers.

REFERENCES

ADHIKARI, R.; VERMA, G.; KHANDELWAL, I.; A Model Ranking Based Selective Ensemble Approach for Time Series Forecasting. Procedia Computer Science, vol. 48, p. 14–21, 2015.

BERGSTRA J. and BENGIO, Y., Random search for hyper-parameter optimization, The Journal of Machine Learning Research, vol. 13, p. 281-305, 2012.

BISHOP, C. M. Pattern recognition and machine learning. New York: Springer, 2006.

BOX, G. E. and JENKINS, G. M. Time series analysis: forecasting and control. Holden-Day, 1970.

BREIMAN, L. Randomizing Outputs to Increase Prediction Accuracy. Machine Learning 40, 229–242, 2000.

BREIMAN, L. Stacked Regressions. Machine Learning 24, 49–64, 1996.

CAO, L.J.; TAY, F.E.H. Support vector machine with adaptive parameters in financial time series forecasting. IEEE Transactions on Neural Networks, vol. 14, no. 6, p. 1506–1518, Nov. 2003.

CHATFIELD, C. Time-series forecasting. CRC Press, 2000.

CHEN C., LIU Y., SUN X., CAIRANO-GILFEDDER, C. D., and TITMUS S. Automobile Maintenance Modelling Using gcForest. IEEE 16th International Conference on Automation Science and Engineering (CASE), pp. 600-605, 2020.

CHEN, K.; WANG, C. Support vector regression with genetic algorithms in forecasting tourism demand. Tourism Management, vol. 28, no. 1, p. 215–226, Feb. 2005

CHIMMULA, V. K. R.; ZHANG, L.. Time series forecasting of COVID-19 transmission in Canada using LSTM networks. Chaos, Solitons & Fractals, vol. 135, p. 109864, Jun. 2020.

CONNEAU, A.; SCHWENK, H.; BARRAULT, L.; LECUN, Y.. Very Deep Convolutional Networks for Text Classification. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, 2017.

DE MATTOS NETO, P.S.G.; MADEIRO, F.; FERREIRA, T.A.E.; CAVALCANTI, G.D.C. Hybrid intelligent system for air quality forecasting using phase adjustment. Engineering Applications of Artificial Intelligence, vol. 32, p. 185–191, Jun. 2014.

DE O. SANTOS JÚNIOR, D.S.; DE OLIVEIRA, J.F.L.; DE MATTOS NETO, P.S.G. An intelligent hybridization of ARIMA with machine learning models for time series forecasting. Knowledge-Based Systems, vol. 175, p. 72–86, Jul. 2019.

DIETTERICH, T.G. Machine Learning for Sequential Data: A Review. Lecture Notes in Computer Science. [S. l.]: Springer Berlin Heidelberg, p. 15–30, 2002.

GOODFELLOW, I., BENGIO, Y., COURVILLE, A. Deep learning (Vol. 1, No. 2). Cambridge: MIT press, 2016.

HOCHREITER, S.; SCHMIDHUBER, J.. Long Short-Term Memory. Neural Computation, vol. 9, no. 8, p. 1735–1780, 1 Nov. 1997.

JONES, S.S.; THOMAS, A.; EVANS, R. S.; WELCH, S.J.; HAUG, P.J.; SNOW, G.L. Forecasting Daily Patient Volumes in the Emergency Department. Academic Emergency Medicine, vol. 15, no. 2, p. 159–170, Feb. 2008.

KIM, K.. Financial time series forecasting using support vector machines. Neurocomputing, vol. 55, no. 1–2, p. 307–319, Sep. 2003.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: PEREIRA, F. et al. Advances in Neural Information Processing Systems 25. Curran Associates, Inc.,. p. 1097--1105, 2012.

LECUN, Y. Generalization and network design strategies. Technical ReportCRG-TR-89-4, University of Toronto, 1989.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, vol. 86, no. 11, p. 2278–2324, 1998.

LI, W.; FU, H.; YU, L.; CRACKNELL, A.. Deep Learning Based Oil Palm Tree Detection and Counting for High-Resolution Remote Sensing Images. Remote Sensing, vol. 9, no. 1, p. 22, 30 Dec. 2016.

LICHMAN M. Uci Machine Learning Repository. 2013. https://archive.ics.uci.edu/ml/

LIPTON, Z. C. The Mythos of Model Interpretability. Queue, vol. 16, no. 3, p. 31–57, Jun. 2018.

MAAS, A. L.; DALY, R. E.; PHAM, P. T.; et al. Learning Word Vectors for Sentiment Analysis. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1. Portland, Oregon: Association for Computational Linguistics, p. 142--150, 2011.

MULLER, A. C; GUIDO, S. Introduction to machine learning with python: A guide for data scientists. 1. ed. Burlington, MA, United States: O'Reilly Media, Inc, USA, 2016.

NAGESH KUMAR, D.; SRINIVASA RAJU, K.; SATHISH, T. River Flow Forecasting using Recurrent Neural Networks. Water Resources Management, vol. 18, no. 2, p. 143–161, Apr. 2004.

OLAH, C. Understanding LSTM Networks. Colah's blog, 2015. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ Last accessed: 10 Nov. 2020.

OLIVEIRA, M. and TORGO, L.. "Ensembles for Time Series Forecasting." ACML, 2014.

ONCHAROEN, P.; VATEEKUL, P.. Deep Learning for Stock Market Prediction Using Event Embedding and Technical Indicators. 5th International Conference on Advanced Informatics: Concept Theory and Applications IEEE, Aug. 2018.

PANG, M.; TING, K.; ZHAO, P.; ZHOU, Z.. Improving Deep Forest by Confidence Screening. IEEE International Conference on Data Mining (ICDM), Nov. 2018.

PEDREGOSA, F., VAROQUAX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, D., BRUCHER, M., PERROT, M., and DUCHESNAY, E.. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, p.2825–2830, 2011.

QIU, X., ZHANG, L., REN, Y., SUGANTHAN, P., & AMARATUNGA, G. Ensemble deep learning for regression and time series forecasting. IEEE Symposium on Computational Intelligence in Ensemble Learning, Dec. 2014.

QIU, X.; ZHANG, L.; NAGARATNAM SUGANTHAN, P.; AMARATUNGA, G. A.J. Oblique random forest ensemble via Least Square Estimation for time series forecasting. Information Sciences, vol. 420, p. 249–262, Dec. 2017.

QUANG, D.; CHEN, Y.; XIE, X.. DANN: a deep learning approach for annotating the pathogenicity of genetic variants. Bioinformatics, vol. 31, no. 5, p. 761–763, 22 Oct. 2014.

R Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria, 2020. Available at: http://www.R-project.org/.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. Nature, vol. 323, no. 6088, p. 533–536, Oct. 1986.

SEZER,O. B.; GUDELEK, M. U.; OZBAYOGLU, A. M.. Financial time series forecasting with deep learning : A systematic literature review: 2005–2019. Applied Soft Computing, vol. 90, p. 106181, May 2020.

SIAMI-NAMINI, S.; TAVAKOLI, N.; SIAMI NAMIN, A.. A Comparison of ARIMA and LSTM in Forecasting Time Series. IEEE International Conference on Machine Learning and Applications (ICMLA), Dec. 2018.

SILVA, E. G.; DE O. JUNIOR, D.S.; CAVALCANTI, G.D. C.; DE MATTOS NETO, P.S. G. Improving the accuracy of intelligent forecasting models using the Perturbation Theory. 2018 International Joint Conference on Neural Networks (IJCNN), Jul. 2018.

TAY, F. E.H; CAO, L. Application of support vector machines in financial time series forecasting. Omega, vol. 29, no. 4, p. 309–317, Aug. 2001.

TORRES, J.F.; GALICIA, A.; TRONCOSO, A.; MARTÍNEZ-ÁLVAREZ, F. A scalable approach based on deep learning for big data time series forecasting. Integrated Computer-Aided Engineering, NL, vol. 25, no. 4, p. 335–348, 5 Sep. 2018.

TSITSIKA, E. V; MARAVELIAS, C. D; HARALABOUS, J. Modeling and forecasting pelagic fish production using univariate and multivariate ARIMA models. Fisheries Science, vol. 73, no. 5, p. 979–988, Oct. 2007.

WU T., ZHAO Y., LIU L, LI H., XU W., and CHEN C.; A novel hierarchical regression approach for human facial age estimation based on deep forest. IEEE 15th International Conference on Networking, Sensing and Control (ICNSC), 2018.

WU, Y., SCHUSTER, M., CHEN, Z., LE, Q. V., NOROUZI, M., MACHEREY, W., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144, 2016.

YIN, L.; SUN, Z.; GAO, F.; LIU, H.. Deep Forest Regression for Short-Term Load Forecasting of Power Systems. IEEE Access, vol. 8, p. 49090–49099, 2020.

YONA, A.; SENJYU, T.; SABER, A.Y.; FUNABASHI T.; SEKINE, H.; KIM, C.; Application of neural network to 24-hour-ahead generating power forecasting for PV system. Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, IEEE, Jul. 2008.

ZHANG, G.P. Time series forecasting using a hybrid ARIMA and neural network model. Neurocomputing, vol. 50, p. 159–175, Jan. 2003.

ZHOU, Z.; FENG, J.. Deep Forest: Towards An Alternative to Deep Neural Networks. Proceedings of the Twenty-Sixth International Joint Conferences on Artificial Intelligence Organization, Aug. 2017.

ZHOU, Z.; FENG, J.. Deep forest. National Science Review, vol. 6, no. 1, p. 74–86, 8 Oct. 2019.

ZHOU, Z.. Ensemble Methods: Foundations and Algorithms. Chapman and Hall/CRC, 2012.

APPENDIX A - RESULTS FOR ALL MODELS

Table 14 - Complete results for the Canadian Lynx time series

Model	MSE	MSE Std	MAE	MAE Std	MAPE	MAPE Std
Original Cascade	2.52E-02	6.71E-04	1.20E-01	2.56E-03	23.70	1.05
Cascade All + LR	7.50E-03	3.08E-04	6.23E-02	2.35E-03	15.53	0.80
Cascade All	1.08E-02	9.76E-04	7.59E-02	4.73E-03	17.98	1.28
Cascade All + Extra	1.22E-02	1.16E-03	7.94E-02	4.90E-03	19.54	1.30
Cascade All + RF	1.29E-02	1.76E-03	8.21E-02	6.25E-03	20.40	1.39
Cascade All + SVR	9.84E-03	-	8.24E-02	-	18.87	-
Original gcForest	4.01E-02	1.02E-03	1.81E-01	2.14E-03	41.23	0.79
Cascade SVR	7.37E-03	-	6.35E-02	-	15.46	-
Cascade SVR + Extra	1.01E-2	1.05E-3	6.9E-2	4.99E-3	17.70	1.55
Cascade SVR + LR	7.02E-03	-	6.02E-02	-	14.59	-
Cascade SVR + RF	1.18E-02	1.55E-04	8.01E-02	6.75E-04	19.69	0.21
Cascade SVR + SVR	7.24E-03	-	5.93E-02	-	14.57	-
Cascade Trees	1.12E-02	1.25E-03	7.92E-02	5.03E-03	18.17	1.51
Cascade Trees + Extra	1.59E-02	1.79E-03	9.49E-02	5.78E-03	23.84	1.50
Cascade Trees + LR	9.88E-03	5.90E-04	7.30E-02	3.10E-03	16.08	0.94
Cascade Trees + RF	1.38E-02	1.39E-03	8.95E-02	4.99E-03	22.73	1.53
Cascade Trees + SVR	1.38E-02	1.73E-03	8.81E-02	6.37E-03	21.34	1.12
MLP	1.43E-02	8.69E-03	9.90E-02	2.15E-02	21.93	6.14
Random Forest	9.99E-03	4.75E-04	7.66E-02	2.06E-03	17.81	0.59
SVR	7.39E-03	-	6.35E-02	-	15.92	-

Source: Author, 2020.

Table 15 - Complete results for the Star Brightness time series

Model	MSE	MSE Std	MAE	MAE Std	MAPE	MAPE Std
Original Cascade	5.90E-04	2.14E-05	1.81E-02	2.30E-04	5.61	0.07
Cascade All + LR	2.51E-04	1.13E-05	1.28E-02	2.71E-04	4.27	0.12
Cascade All	4.17E-04	1.79E-05	1.52E-02	2.92E-04	4.89	0.20
Cascade All + Extra	3.11E-04	8.88E-06	1.39E-02	2.38E-04	4.96	0.09
Cascade All + RF	2.90E-04	8.30E-06	1.32E-02	2.25E-04	4.51	0.08
Cascade All + SVR	1.31E-03	6.88E-05	2.85E-02	9.69E-04	11.54	0.24
Original gcForest	1.46E-02	1.10E-04	9.59E-02	3.36E-04	27.84	0.13
Cascade SVR	1.96E-03	-	3.71E-02	-	15.47	-
Cascade SVR + Extra	3.10E-4	2.79E-6	1.37E-2	8.83E-5	4.38	0.05
Cascade SVR + LR	1.42E-04	-	9.88E-03	-	3.35	-
Cascade SVR + RF	2.92E-04	4.85E-06	1.34E-02	1.34E-04	4.48	0.07
Cascade SVR + SVR	1.72E-03	-	3.41E-02	-	12.61	-
Cascade Trees	3.35E-04	1.08E-05	1.41E-02	3.13E-04	4.78	0.14
Cascade Trees + Extra	3.32E-04	9.47E-06	1.39E-02	2.70E-04	4.68	0.13
Cascade Trees + LR	3.54E-04	1.66E-05	1.38E-02	4.32E-04	4.62	0.14
Cascade Trees + RF	3.59E-04	1.84E-05	1.27E-02	5.36E-04	4.31	0.21
Cascade Trees + SVR	1.28E-03	5.33E-05	2.89E-02	7.17E-04	11.44	0.15
MLP	1.73E-04	1.41E-05	1.10E-02	3.41E-04	3.78	0.26
Random Forest	3.42E-04	9.36E-06	1.50E-02	2.62E-04	5.14	0.10
SVR	1.81E-03	-	3.47E-02	-	10.62	-

Source: Author, 2020.

Table 16 - Complete results for the Sunspot time series

Model	MSE	MSE Std	MAE	MAE Std	MAPE	MAPE Std
Original Cascade	1.70E-02	2.27E-04	9.17E-02	4.50E-04	34.98	0.23
Cascade All + LR	2.33E-02	6.64E-04	1.03E-01	1.43E-03	41.00	0.63
Cascade All	1.50E-02	6.67E-04	8.30E-02	1.86E-03	29.49	0.75
Cascade All + Extra	2.05E-02	1.01E-03	9.79E-02	1.77E-03	34.35	0.99
Cascade All + RF	1.63E-02	7.80E-04	8.77E-02	2.36E-03	31.92	1.11
Cascade All + SVR	2.08E-02	1.14E-03	1.03E-01	2.19E-03	51.53	1.03
Original gcForest	4.83E-02	3.96E-04	1.64E-01	6.52E-04	80.36	0.48
Cascade SVR	9.56E-03	-	7.60E-02	-	42.29	-
Cascade SVR + Extra	1.49E-2	2.14E-4	8.85E-2	5.50E-4	34.36	0.31
Cascade SVR + LR	1.95E-02	-	9.35E-02	-	32.54	-
Cascade SVR + RF	1.28E-02	5.14E-04	8.27E-02	1.62E-03	33.35	1.05
Cascade SVR + SVR	1.18E-02	-	7.99E-02	-	40.53	-
Cascade Trees	1.92E-02	9.91E-04	9.61E-02	2.30E-03	34.51	1.24
Cascade Trees + Extra	1.61E-02	6.01E-04	8.57E-02	1.75E-03	30.05	0.95
Cascade Trees + LR	1.85E-02	8.41E-04	9.46E-02	2.42E-03	33.45	1.35
Cascade Trees + RF	2.12E-02	8.78E-04	1.01E-01	2.88E-03	37.38	1.75
Cascade Trees + SVR	1.60E-02	1.09E-03	9.10E-02	2.43E-03	45.95	0.88
MLP	1.18E-02	8.63E-04	8.16E-02	3.77E-03	34.37	2.29
Random Forest	1.91E-02	3.19E-04	9.53E-02	9.71E-04	33.59	0.61
SVR	1.48E-02	-	8.77E-02	-	43.60	-

Source: Author, 2020.

Table 17 - Complete results for the S&P 500 time series

Model	MSE	MSE Std	MAE	MAE Std	MAPE	MAPE Std
Original Cascade	3.79E-02	8.45E-06	1.59E-01	2.60E-05	17.72	0.00
Cascade All + LR	1.51E-03	-	3.06E-02	-	3.40	-
Cascade All	2.18E-02	5.99E-05	1.32E-01	2.24E-04	15.14	0.03
Cascade All + Extra	3.77E-02	-	1.59E-01	-	17.64	-
Cascade All + RF	3.80E-02	3.90E-04	1.60E-01	1.21E-03	17.77	0.15
Cascade All + SVR	1.20E-02	-	1.07E-01	-	12.80	-
Original gcForest	3.78E-02	6.62E-06	1.59E-01	2.03E-05	17.68	0.00
Cascade SVR	1.51E-02	-	1.20E-01	-	14.34	-
Cascade SVR + Extra	4.00E-2	9.17E-7	1.66E-1	2.76E-6	18.51	0.00
Cascade SVR + LR	1.37E-04	-	9.82E-03	-	1.17	-
Cascade SVR + RF	3.80E-02	2.85E-04	1.60E-01	8.81E-04	17.74	0.11
Cascade SVR + SVR	1.22E-02	-	1.08E-01	-	12.90	-
Cascade Trees	3.80E-02	9.11E-05	1.60E-01	2.83E-04	17.76	0.03
Cascade Trees + Extra	3.77E-02	5.37E-05	1.59E-01	1.65E-04	17.66	0.02
Cascade Trees + LR	3.76E-02	1.04E-04	1.59E-01	3.20E-04	17.62	0.04
Cascade Trees + RF	3.80E-02	4.31E-04	1.60E-01	1.33E-03	17.75	0.16
Cascade Trees + SVR	7.95E-02	2.75E-07	2.59E-01	5.32E-07	29.93	0.00
MLP	2.25E-04	1.56E-04	1.15E-02	3.53E-03	1.35	0.39
Random Forest	3.85E-02	4.24E-04	1.61E-01	1.30E-03	17.94	0.16
SVR	1.16E-02	-	1.05E-01	-	12.57	-

Source: Author, 2020.

APPENDIX B - PARAMETER CONFIGURATION FOR THE BEST MODELS

For the Canadian Lynx time series, the best performing model was the Cascade SVR+LR, which performed better for two out of the three metrics. Table 18 shows the best parameter configuration for this model, which was made up of two SVRs. The parameters for the Linear Regression are the default ones for the library scikit-learn.

Table 18 - Parameters for the best Cascade SVR+LR model for Canadian Lynx.

Model	Parameters	Values
SVR 1	Gamma	0.01
	C	0.01
SVR 2	Gamma	0.1
	C	100

Source: Author, 2020. Parameters for the linear regression are the default one from the library scikit-learn.

For the Sunspot time series, the best model was the Cascade SVr for two out of the three analyzed metrics. Like the model for the Canadian Lynx, the best configuration was made up of two SVRs, but combined with the mean instead. Table 19 shows the parameters for these two models.

Table 19 - Parameters for the best Cascade SVR+LR model for Sunspot.

Model	Parameters	Values
SVR 1	Gamma	0.0001
	C	1000
SVR 2	Gamma	1
	C	100

Source: Author, 2020

For the Star Brightness dataset, the best performing model was also the Cascade SVR+LR, which showed the best performance for all three metrics. This time, the best configuration was made up of 6 SVRs models, the maximum amount considered in the experiments. Table 20 shows the best parameters for all SVRs.

Table 20 - Parameters for the best Cascade SVR+LR model for Star Brightness time series.

Model	Parameters	Values
SVR 1	Gamma	0.001
	C	1000
SVR 2	Gamma	0.1
	C	0.1
SVR 3	Gamma	0.01
	C	1
SVR 4	Gamma	0.001
	C	1
SVR 5	Gamma	0.0001
	C	100
SVR 6	Gamma	0.001
	C	0.01

Source: Author, 2020.

For the S&P 500 dataset, there was not a single model with a clear best performance. The MLP and the Cascade SVR+LR had similar performances for this time series. Table 21 shows the best configuration of parameters for the MLP while Table 22 shows the best configuration for the Cascade SVR+LR algorithm. While the MLP is a single model, the Cascade SVR+LR is composed of five SVRs

Table 21 - Parameters for the best MLP model for the S&P 500 time series.

Model	Parameter	Value
MLP Regressor	Initial Learning Rate	0.001
	Activation Function	logistic
	Nodes in Layer	50
	Solver	lbfgs
	Max Iterations	100000

Source: Author, 2020.

Table 22 - Parameters for the best Cascade SVR+LR model for S&P 500.

Model	**Parameters**	**Values**
SVR 1	Gamma	0.001
	C	1000
SVR 2	Gamma	0.001
	C	1000
SVR 3	Gamma	0.01
	C	1
SVR 4	Gamma	0.1
	C	100
SVR 5	Gamma	0.001
	C	0.1

Source: Author, 2020.