VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

MODELLING AND DATA ANALYSIS MASTER'S STUDY PROGRAMME

Master's thesis

Forecasting Nonstationary and Nearly Nonstationary Time Series Using Machine Learning Methods

Nestacionarių ir beveik nestacionarių laiko eilučių prognozavimas naudojant mašininio mokymosi metodus

Reda Vaičiūnaitė

Supervisor: doc. dr. Jurgita Markevičiūtė

Vilnius, 2021

Forecasting Nonstationary and Nearly Nonstationary Time Series Using Machine Learning Methods

Abstract

The purpose of the study is to examine whether the differences between nonstationary and nearly nonstationary time series have a significant impact on forecasting using machine learning models. One of the main objectives is to verify the ability of the machine learning models to make quite accurate one-step ahead predictions for both nonstationary and nearly nonstationary time series. Another objective is to compare the forecasting performance of machine learning and traditional statistical models. For the analysis three machine learning models, including the Multilayer Perceptron (MLP) network, Recurrent Neural Network (RNN) and Support Vector Regression (SVR), and a single traditional statistical method Autoregressive Integrated Moving Average (ARIMA) are used. This work consists of the simulation study, including the first order autoregressive time series case analysis, and the application to the real world data with an example of financial market time series. The results show that in most cases machine learning models predict both nonstationary and nearly nonstationary time series quite accurately. However, machine learning models are not able to make significantly better predictions for the time series, which follow a random walk, in comparison to the traditional statistical methods.

Key words : nonstationary time series, nearly nonstationary time series, machine learning methods, statistical time series forecasting models, time series forecasting

Nestacionarių ir beveik nestacionarių laiko eilučių prognozavimas naudojant mašininio mokymosi metodus

Santrauka

Šio tyrimo tikslas yra ištirti, ar skirtumai tarp nestacionarių ir beveik nestacionarių laiko eilučių yra reikšmingi prognozavimui, naudojant mašininio mokymosi metodus. Vienas iš pagrindinių uždavinių yra patikrinti, ar mašininio mokymosi metodai gali pateikti gana tikslią vieno žingsnio į priekį prognozę tiek nestacionarioms, tiek beveik nestacionarioms laiko eilutėms. Kitas uždavinys yra palyginti mašininio mokymosi bei tradicinių statistinių modelių prognozavimo tikslumą. Tyrime taikomi trys mašininio mokymosi modeliai: daugiasluoksnio perceptrono neuroninis tinklas (MLP), rekurentinis neuroninis tinklas (RNN) ir palaikančiųjų vektorių regresijos modelis (SVR), bei vienas tradicinis statistinis modelis - autoregresinis integruotas slenkamųjų vidurkių (ARIMA) metodas. Šį darbą sudaro simuliacinis tyrimas, kuriame naudojama pirmojo laipsnio autoregresinė laiko eilutė, bei modelių pritaikymas laiko eilutei, sudarytai iš realių finansų rinkos duomenų. Remiantis rezultatais, daugeliu atvejų, mašininio mokymosi metodai tiek nestacionarias, tiek beveik nestacionarias laiko eilutes prognozuoja gana tiksliai. Tačiau, laiko eilutėms, kurių trajektorijos panašios į atsitiktinį klaidžiojimą, mašininio mokymosi modeliai nepateikia reikšmingai geresnių prognozių lyginant juos su tradiciniais statistiniais metodais.

Raktiniai žodžiai : nestacionarios laiko eilutės, beveik nestacionarios laiko eilutės, mašininio mokymosi metodai, statistiniai laiko eilučių prognozavimo modeliai, laiko eilučių prognozavimas

Contents

Introduction

The efficient forecasting of time series plays an important role in various fields. In many real world situations, the ability to anticipate the future is very essential and necessary, for instance, to businesses it helps to set goals, properly allocate their budget or make important financial decisions and adjustments in strategy required to avoid a decrease in profit, for investors forecasting of future earnings assists to make profitable investments, in meteorology a knowledge about the weather in the future makes it possible to foresee natural disasters and there are only a few examples where forecasting has a major impact. For a long time, various statistical models have been used in an attempt to get predictions of the observations in different areas under consideration. However, the efforts to find the model which would be efficient for every time series forecasting were unsuccessful due to the various characteristics of series. It was found that very huge influence in the efficiency of forecasting and selection of the appropriate model has the stationarity of time series. The presence of persistence met in a major part of the real world time series usually makes many difficulties in forecasting due to the randomness in their behaviour [38, 55]. The high degree of persistence is related with the presence of the unit roots in time series or their, for example, autoregressive, representations and that indicates the nonstationarity of time series [38, 55]. Usually, the time series with a high degree of persistence and the unit roots very close to unity, which are called as nearly nonstationary, have more similar properties and tendencies in the behaviour to nonstationary time series than stationary ones. Such time series in practical examples often are simply treated as nonstationary.

A great number of investigations have been performed to analyze the nonstationary time series by trying to find the model which would be able to make most accurate predictions in various forecasting problems and it remains the area of interest. After when the machine learning models have been proposed and applied for the time series forecasting it was noticed that they are able to deal with a complex behaviour of the nonstationary time series and show a quite well forecasting performance in most cases, sometimes even outperform traditional statistical models. Many related works could be found with the applications of machine learning models for the different time series forecasting, however, to our knowledge, no prior studies have examined what effect the changes in the degree of persistence in time series has on forecasting using machine learning models, therefore this is the main interest of this work.

The analysis of the first order autoregressive time series (AR(1)) case offers an appropriate framework on which to test whether there are some significant differences in the nonstationary and nearly nonstationary time series forecasting using machine learning models. Therefore, the simulation study is performed in this work to apply several of the most commonly used machine learning models and examine their ability to make one-step predictions, which are of significant importance in various applications, for both nonstationary and nearly nonstationary time series. Three machine learning models are chosen in the experimental part of this study: Multilayer Perceptron (MLP) network, Recurrent Neural Network (RNN) and Support Vector Regression (SVR). The traditional statistical autoregressive integrated moving average (ARIMA) model is included in the experiment as a benchmark and the comparison of the forecasting performance of models is presented. There could be formulated two hypotheses which are required to be approved or denied in this work. One of them is that machine learning models could achieve similarly well performance on both nonstationary and nearly nonstationary time series forecasting. Another is that machine learning models could outperform traditional statistical forecasting models on the nonstationary and nearly nonstationary time series forecasting.

In addition, the effect of the changes in the value of the AR(1) coefficient for the forecasting performance of models is examined and the real world application is presented. Due to the fact that time series of the financial markets usually have a high degree of persistence, in this experiment forecasting performance of models is tested and compared using the Offset Market Exchange Gross

Index of Vilnius (OMX Vilnius GI) data.

The rest of the work is organized as follows. In Chapter 1, the literature review is presented, including the discussion of several various traditional statistical and machine learning models, their applications for the nonstationary real world time series forecasting with their strengths and weaknesses in making predictions. Additionally, some works related to the findings of the similarities between nonstationary and nearly nonstationary time series are reviewed. In Chapter 2 the main definitions and theoretical concepts, required for the experimental part of this work, are discussed, including the nonstationary and nearly nonstationary AR(1) processes, selected statistical (ARIMA) and machine learning (MLP, RNN and SVR) models and forecasting performance metrics. In Chapter 3, simulation study and the real world data application of the time series forecasting are explored using models described in Chapter 2. Final conclusions are given in Chapter 4.

1 Literature review

In the real world, it is very important to observe how variables change over time. The set of observations collected at certain time intervals is referred to as time series data [36]. Time series analysis has become very important in many fields including economics, medicine, meteorology, engineering, politics, and others. One of the most important and widely discussed topics in time series analysis is a prediction of future values. Analyzing sequential observations it was noticed that values in the future usually depend on the present or previous observations. Such dependence encouraged deeper analysis of time series and first attempts to predict future events from the occurred data. Very significant was an introduction to autoregressive (AR) models by G.U. Yule in 1926 [44]. The autoregressive model represents an opportunity to forecast the variable of interest using a linear combination of some number of historical measurements. After successful supplement by E. Slutsky who in 1937 presented Moving Average (MA) schemes and proposed to use a weighted linear sum of last forecasting errors for future value prediction, H. Wold (1938) decided to combine these models into an autoregressive moving average (ARMA) model [44]. It is important to emphasize that using ARMA model for modeling real-life time series requires time series to be stationary for making precise predictions. Time series whose statistical properties, such as mean, variance and autocorrelation do not depend on time are defined as stationary. Nevertheless, in the real world nonstationary time series, whose statistical properties change over time, are more common and forecasting models based on the assumption of stationarity are using after applying some mathematical transformations for these time series. One basic transformation is to compute the differences between consecutive observations, better known as differencing. This method helps to eliminate or reduce trend and seasonality and stabilize the mean and variance of a time series. G. E. P. Box and G. Jenkins (1970, 1976) popularized the statistical ARMA model by suggesting differencing to make the series stationary. The approach proposed by Box and Jenkins which is known as the Box-Jenkins methodology produced an autoregressive integrated moving average (ARIMA) model. ARIMA models and the Box-Jenkins methodology became one of the most widely used time series forecasting methods in practice. [44] Further developments of time series forecasting address the issues of dealing with the non-constant volatility of the data. R. F. Engle (1982) introduced to an autoregressive conditional heteroscedastic (ARCH) model which has been widely used to model volatility of time series and was developed to better account for a non-constant behavior in comparison to ARIMA models [23, 63]. ARCH model assumes that the conditional (not fixed over time, changes depending on the available data) variance is linearly dependent on one or more last squared values of the time series [12]. After four years T. Bollerslev (1986) suggested the generalized ARCH (GARCH) model which represent a linear relationship of conditional variance against some number of last previous squared time series values and conditional variance values [12, 23]. However, these already mentioned and many other traditional econometric models for time series prediction are established on the assumption that data are stationary. Applying some mathematical transformations for time series, such as differencing mentioned above, is not always efficient. The selection of the suitable transformation is a challenging task and sometimes it is completely ineffective when we choose to investigate time series which fluctuates in a highly nonlinear and nonstationary way. After many attempts to forecast nonstationary time series as accurately as possible, there was noticed that the traditional statistical econometric models cannot exhibit great advantages in processing complex nonlinear and nonstationary data and prediction results may be unsatisfactory [76].

A great breakthrough has been made after proposing machine learning (ML) methods as alternatives to statistical ones for the time series forecasting [69]. Machine learning is an application of Artificial Intelligence (AI) that brings together statistics and computer science to enable computers automatically learn how to do a given task and improve from experience without being programmed to do so.

One of the main machine learning based techniques is Neural Networks (NN) which proved to be very suitable to pattern recognition, nonlinear signal processing, classification and also time series forecasting, especially if they are nonstationary and nonlinear [3, 6]. F. E. H. Tay and L. Cao in their paper (2001) represent NN as "universal function approximators that can map any nonlinear function without a priori assumptions about the properties of the data" [65]. There is a wide choice of different NN algorithms available in the literature.

Earliest delivered is an Artificial Neural Network (ANN), which works similar to the human brain. ANN try to recognize regularities and patterns in the input data, involve understanding and learning from experience until finally, they are able to provide generalized results based on the acquired knowledge [3]. The simplest ones are feedforward ANN (FANN) and their probably most widely used form is the multilayer perceptron (MLP), derived in 1960 [30]. MLP network (MLPN) presents a nonlinear functional mapping from input nodes, which are the past observations of the time series, to the output nodes – predicted future values [3]. To perform a nonlinear mapping between input and output layers this network uses some number of hidden nodes/neurons placed into one or more layers. Trying to get accurate precisions it is important to consider an optimal ANN architecture by specifying the number of nodes and hidden layers and finding the optimal values for the connection weights [30]. Hidden nodes are used to increase the flexibility of the model, nevertheless, an inappropriate network size can cause contrary results. An insufficient number of hidden nodes leads to the inability of a neural network to learn data and solve the problem, whilst too many hidden neurons transform learning and generalization into memorizing data, which could be the reason of an overfitting, long and unnecessary training time, difficult implementation of the model and falling into local minima [30]. I. Kaastra and M. Boyd in their paper (1995) [33] provide a very informative practical introductory guide in the design of a neural network for forecasting economic time series data and affirm that "a neural network with one hidden layer with a sufficient number of hidden neurons is capable of approximating any continuous function". Also, they summarize several suggestions for selecting the optimal number of hidden neurons for three-layer network proposed by different researchers: use the geometric pyramid rule (Masters), which specifies that the number of hidden neurons is equal to the square root of the product of the number of input and output neurons or ranges from one-half to two times this value, choose the number of hidden nodes equal to 75% (D. Baily and D. M. Thompson) of or between one-half to three times (J. O. Katz) the number of input neurons or double the number of hidden neurons until the network's performance on the testing set deteriorates (O. Ersoy). Authors summarize, that optimal number of hidden neurons could be found by experimentation such as training a group of NN with different numbers of hidden neurons and choosing the one which has the least error or changing the number by adding or removing hidden nodes during training until network performance starts deteriorating. From this paper and many others, it is obvious that there is no general rule for the selection of an appropriate number of hidden layers and nodes because what is appropriate for one time series may not be at all suitable for others due to their various characteristics. Besides the selection of network size, another main problem of constructing an optimal ANN architecture is to find the optimal values of connection weights. That could be done using the training algorithm and several of them are mentioned in [30]: Quick-Prop (QP), Orthogonal Least Square (OLS), Levemberg-Marquart (LM), Resilient Propagation (RPROP) and Back Propagation Algorithm (BP), by emphasizing widely used BP method, first introduced by Werbos in 1974 for three layer perceptron network. BP algorithm helps to reduce errors of prediction by adjusting network biases and weights into those that minimize the error function using a gradient descent technique [30]. After the successful introduction of this algorithm many variations have been proposed, such as Quick Back Propagation (QBP), Resilient Back Propagation (RBP), Broyden – Fletcher – Goldfarb – Shanno Quasi-Newton Back Propagation (BFGS) [54]. Previous studies have shown, that ANN with a BP training algorithm often outperforms traditional econometric forecasting models analyzing nonstationary time series with irregular fluctuations and accidental pattern changes due to "its unique non-parametric, non-assumable, noise-tolerant and adaptive properties" [34]. J. Kamruzzaman and R. A Sarker (2003) used three ANN based forecasting models: standard Back Propagation (BP), Scaled Conjugate Gradient Algorithm (SCG) and Back Propagation with Regularization (BPR) for predicting currency exchange rates of Australian Dollar with six other currencies and showed that all of them outperformed the traditional ARIMA model [34]. The same conclusion was made by Dr. S. K. Safi (2013) who predicted electricity consumption in Gaza Strip by comparing simple ANN (MLPs using a BP algorithm) and ARIMA models [54]. Authors in [15] propose an informative comparison of neural networks and conditional heteroscedastic models like ARCH, GARCH, GARCH in mean (GARCH-M), threshold GARCH (TGARCH), exponential GARCH (EGARCH) and integrated GARCH (IGARCH) for forecasting the exchange rate time series, by demonstrating that NN models have better performance and can be effectively employed even to estimate the volatility. In this paper two networks are used: already widely discussed MLP and Radial Basis Function Network (RBFN), where the second one gives superior forecasting results due to better extraction of the information necessary to perform a good generalization from the training set.

RBFN has a similar architecture to MLP but there are some differences: RBFN cannot have more than one hidden layer, is often easier to be trained than MLP and the hidden nodes use radial basis functions (RBF) as the identity activation functions [15, 70]. Input and hidden nodes in RBFN are connected directly without using weights, each hidden node contains RBF and the weighted sum of the outputs of RBF is equal to the predicted value. RBFN acts as local approximation networks while MLPN works globally due to the fact that the outputs in RBFN are determined by specified hidden nodes in certain local receptive fields and in MLPN they are decided by all the neurons [70]. RBF gives an output depending on the distance between incoming variables and center positions and on the width of the RBF unit [53]. It is obvious, that RBFN has few additional parameters, compared to MLP, that have to be determined. The parameters of RBF units are established in three steps of the training activity: firstly, using some form of clustering algorithm the unit centers are determined, after that, the widths are specified by the nearest-neighbour method and finally multiple linear regression techniques are used to calculate the weights connecting the RBF units and output nodes [70]. More detailed information about the determination of parameters of the RBF and a comparison between RBFN and BP models used to the short-term system load forecasting is presented in [53]. The results of this research show that the RBFN better provide peak and total load forecasts for the next day, has a shorter training time than BP and can also compute, at no additional computational cost, reliability measures such as an extrapolation index and a confidence interval for the forecast, which is an expression of local goodness of fit. Although this research demonstrates that very good predictions can be obtained using Gaussian function as an activation function, in [27] it is declared that the choice of basis function strongly depends on the data set and it is useful to evaluate the effects of more recognized basis functions suitable for RBF networks, such as multiquadratic, inverse multiquadratic, thin plane spline, cubic or linear basis function. Authors in [60] propose to incorporate Bayesian regularization (BR) method into RBFN and introduce the Bayesian regularized RBFN (BR-RBFN) model. Bayesian regularization converts a nonlinear regression into a "well-posed" statistical problem in the manner of a ridge regression and helps to make neural network more robust [41]. Using regularization methods helps to penalize the network complexity and to avoid or reduce the risk of network overfitting and overtraining. The research in [60] shows that BR-RBFN is able to effectively capture the nonlinear behaviour of daily stock return and perform well with prediction accuracy.

In traditional NN all input (and output) variables are assumed to be independent of each other but it was noticed that sometimes incorporating the sequence dependency could be very useful in forecasting. This idea was implemented by introducing the Recurrent NN (RNN), which is well known due to memorizing its previous computations and applying them in the subsequent forecasting steps. Differently from other NN, RNN uses not only input data but also the historical information of the previous outputs for making further prediction. Although RNN is hard to train and the problem of vanishing gradient or exploding gradient may appear, this model quickly became a competitive forecasting method [28, 50, 69]. Authors of [11] demonstrated that RNN could be used for the short-term forecasting of wind speed and provide better predictions than do standard linear ARIMA models. An additional suggestion was proposed in this paper to incorporate other measurable covariates which could provide a significant forecast improvement. A comparative study of the performance of ARIMA, multi-layer feed-forward NN (MFNN) or otherwise known as MLP with BP and RNN models in predicting compressor failures of a repairable system and detecting reversals was carried out in [29]. The results of this investigation show that MFNN generates poorer forecasts and the lowest percentage of correct reversal detection compared with ARIMA and RNN which both almost perfectly managed with these tasks. However, authors emphasize that the higher long term forecasting errors suggest using ARIMA and RNN for short term predictions. One of the main disadvantages of RNN is that it can remember only a few earlier steps in the sequence [50, 69]. As the solution to this challenging problem was proposed a special kind of RNN called Long Short-Term Memory (LSTM) network, which is very efficient and widely analyzed nowadays. LSTM contains a series of reconnected memory modules and each of them is composed of a cell, which remembers values over arbitrary time intervals, and three gates (input, output and forget) to regulate the flow of information into the cell. Results of the research in [76] show that LSTM can cope perfectly with the complex electric load time series, which poses a great challenge for long-horizon time series forecasting due to the high non-stationarity and non-seasonality, also can make reliable predictions and outperform many other forecasting methods. However, LSTM uses many computational resources and it is really hard to train due to a long training time.

The literature review shows that many different NN methods have been proposed for the non-stationary time series forecasting but, obviously, it is not possible to single out a specific one that is perfectly suitable in all cases. Due to the different weaknesses and strengths of various models, the idea of creating combinations of models was suggested. One of the most widely used is the hybrid ARIMA-ANN model. Previous studies have shown that ARIMA could be a perfect tool for linear predictions while the ANN could not handle both linear and nonlinear patterns equally well but has a good performance in the face of nonlinear problems [43, 73]. The researches in [73] and [43] confirm that the combination of these models can often improve the accuracy of forecasting the real world time series, which usually contain both linear and nonlinear patterns, and can increase the chance to capture different patterns in the data. Another interesting and successful suggestion is to preprocess time series using wavelet transform (WT) before feeding it into the ANN model as input. WT spreads the main time series into subcomponents called wavelets, which are a scaled and shifted version of the fixed function namely the mother wavelet [2, 26]. Two researches in [26] and [2] propose a hybrid model composed of discrete WT (DWT) and widely used ANN type the MLP NN (MLPNN) by calling this combination WTMLPNN and WA-ANN respectively. Authors of these papers based on other related works and their experimental results present WT as an effective tool in analyzing nonstationary time series which helps to provide useful decompositions of the time series, capture useful information from data on various decomposition levels and thus improve the performance of ANN which sometimes has limitations with non-stationary data and requires data preprocessing. Results of researches show that WT combination with ANN is able to provide better forecasting results compared to traditional ANN and ARIMA models. In [37] the hybrid model called the wavelet RBF neural network (WRBFNN) is proposed by comparing its performance with wavelet feed forward neural network (WFFNN) model which is just differently named WT combination with MLPNN. Four time series are considered with different characteristics: Chaotic McGlass data (nonlinear and stationary in mean and variance), Electricity Usage (nonstationary on variance), Traffic Fatalities (nonstationary on mean and variance with linear trend) and Canadian Lynx data (nonstationary on mean and variance with nonlinear trend). Comparative results show that for prediction of nonlinear and stationary data both methods can be used but WRBFNN is superior to WFFNN. However, according to the authors, when applied models to nonstationary data forecasting WRBFNN will achieve higher generalization performance only on time series with a simple pattern and on those with a complex pattern WFFNN is a better choice.

After a successful introduction of NN, V. Vapnik and his co-workers designed the Support Vector Machines (SVM) technique (1995) which, similarly to NN, initially was useful in solving pattern classification problems, such as text or images classification, face identification or handwriting recognition [3]. Later it was noticed that SVM could perform well in time series forecasting due to its remarkable generalization performance, the sparse representation of solution and the ability not to fall into local minima, which is one of the major problems of NN [10]. SVM algorithm applied to regression problems is called Support Vector Regression (SVR). The main idea in SVR is to map data points of the input space into higher dimensional feature space using a nonlinear mapping and to estimate the regression by using a set of linear functions that are defined in this space. [3, 10] This property helps for the quality and complexity of the solution to be independently controlled, irrespective of the dimension of the input space [3]. After data mapping, SVR carries out the regression estimation by minimizing risk, which is measured by the new type of loss function called ϵ-insensitive loss function proposed by Vapnik and tries to find a decision rule with good generalization capacity by using the structural risk minimization (SRM) principle [3, 10]. This unique principle usually helps SVR to achieve higher generalization performance than traditional NN could reach by implementing the empirical risk minimization (ERM) [10]. One of the main advantages of SVR is that linear functions could be used for training in the high dimensional feature space without difficulties just by solving a linearly constrained quadratic optimization problem (QPP) [3]. This feature makes the solution "always unique and globally optimal, unlike other networks' training which requires nonlinear optimization with the danger of getting stuck into local minima" [10]. It should be emphasized that applying SVR to forecasting one of the most important tasks is to choose kernel function which is used for dealing with feature spaces of arbitrary dimensionality without the explicit computation of the nonlinear mapping [10]. Functions that satisfy Mercer's condition could be kernel functions and probably most widely used are linear, polynomial, radial basis function (RBF), Gaussian and neural network (NN) kernels [3, 10]. Like using NN for time series forecasting the essential task is to select optimal values for parameters the same problem remains important by implementing SVR. The good performance of SVR in time series forecasting usually depends on the proper selection of values of kernel parameters, the regularization constant and the tube size [10]. There is no one best method for choosing optimal parameters but some techniques such as cross-validation or Bayesian inference could be used to solve this problem. Authors in [10] use SVR for financial time series forecasting and choose the Gaussian function as the kernel function due to its good performance under general smoothness assumptions, the ability to deal with strongly nonlinear data and better results obtained compared to those that have been received using a polynomial kernel. The optimal values of parameters in this research are those that produce the best result in the validation set. Authors in this paper show that SVR could be perfectly applied in inherently noisy and nonstationary financial time series forecasting, outperform multilayer back-propagation neural network, here shortly called as BP, and perform similar to regularized RBFN due to the ability of SVR and regularized RBFN to minimize the regularized risk function. In this research also an interesting suggestion to use adaptive parameters is proposed. Authors remark that "in nonstationary financial time series, it is usually believed that the information provided by the recent training data points is more important than that provided by the distant training data points" and as the solution, they suggest to use ascending regularization constant and descending tube which helps to place more weights on the recent training data points and less on the distant points. The results of the research show that adaptive parameters proposed by incorporating the nonstationarity of financial time series could help to achieve higher generalization performance in forecasting. Another comparative research was proposed by R. Achanta in [1] for electric load forecasting using SVR with polynomial kernel function and MLP trained with BP algorithm. Obtained results as in [10] show SVR superiority against MLP with BP and it is concluded that SVR with proper selection of parameters could be a good replacement for some of the NN based models for electric load forecasting. However, despite all advantages, SVM based forecasting method requires an enormous amount of computation and a very long training time when the data set, used for training, is large [3].

During the last few decades, many different SVM forecasting algorithms have been proposed. Trying to simplify computations and ensure higher precision in forecasting J. A. K. Suykens and J. Vandewalle suggested a least squares version of SVM (LS-SVM) that transforms the traditional, long and computationally difficult QPP to a simultaneous linear system problem [3, 19, 35]. In LS-SVM the least squares loss function is used to construct the optimization problem based on equality constraints instead of the ϵ-insensitive loss function and inequality constraints used in SVM [35, 66]. LS-SVM used for regression is called LS-SVR [35]. According to the comparison of SVR and LS-SVR made by H. Wang and D. Hu (2005) [66], using LS-SVR for function estimation the sparseness is lost, which is an attractive property of SVR but to get a sparse solution they introduce a simple pruning method. Their research shows that the generalization performance of the LS-SVM with the SVM for regression is comparable but LS-SVR is preferred for large scale regression problems due to its high efficiency solution procedure. An extensive comparative study of the LS-SVR, ARIMA, ANN (MLP using the Levenberg-Marquardt (LM) algorithm) and SVR performance in the streamflow, which is known as very complex and difficult to model time series, forecasting was proposed in [58]. Results of this research show that the performance of the ARIMA is worse in some cases than other AI methods, due to its failure to capture the pattern of extreme values while other methods are able to deal with the nonlinear and highly complex behaviour of the streamflow process. The overall comparison shows that LS-SVR outperforms or obtains similar good predictions as other models for the streamflow forecasting. In [46] LS-SVR is successfully applied for meteorological time series (solar irradiance, wind speed and direction, air temperature, relative humidity, and pressure) a single-step (1 h ahead) prediction and a comparison between LS-SVR, MLPN, RBFN and RNN is proposed for predicting the future values based on the past values of air temperature and pressure. Comparative results show that LS-SVR performs better than other ANN architectures, from which the best predictions were obtained by using RNN and the worst prediction performance achieved by MLPN.

Specially, for dealing with nonstationarity of time series, L. Cao and Q. Gu in [9] proposed a modified version of SVM called Dynamic SVM (DSVM). DSVM used for regression also could be renamed into DSVR. This model suggests using an exponentially increasing regularization constant and an exponentially decreasing tube size to deal with structural changes in the data instead of fixed ones used in standard SVR. Actually, this proposed idea is similar to the selection of adaptive parameters suggested in [10]. Both researches in [10] and [9] show that the dynamic selection of parameters could be very useful in forecasting nonstationary time series. DSVR method is developed by incorporating the idea of discounted least squares (DLS) into SVR, which propose weighting recent training data points more heavily than distant observations used for training. The experiment is made to compare the performance of DSVR and SVR using different simulated and real data sets. The experimental results show that DSVR could be more advantageous and robust method in forecasting nonstationary time series than the SVR. Also, DSVR uses fewer support vectors compared to SVR, resulting in a sparser representation of the solution. However, authors note that exponential weight functions are effective in analyzed cases but the application of other weight functions needs to be more investigated as well as other procedures used for the selection of optimal control rates to control the ascending regularization constant and descending tube.

As the combinations of various models have proven to be very effective in time series forecasting, many hybrid models have been proposed incorporating the models based on SVM. Due to the fact that ARIMA well performs on linear time series predicting while SVM, like ANN, focus much on the nonlinear fitting and cannot accurately forecast the linear basic part of time series, the hybrid model of ARIMA and SVM (ARIMA-SVM) was proposed in [47] for the short-term load forecasting. Authors note that the power load series usually is cyclical, slowly increasing and very sensitive to external factors such as the weather or days of the week. According to the results of a simulation, using ARIMA for the load forecasting, external factors cannot be taken into account and the deviation series obtained is cyclical though the random fluctuation is constant in a short period of time. SVM is used here to extract the sensitive component from the deviation and to improve the prediction accuracy. By adding the forecasting load deviation to the forecasting load higher precision is achieved than using ARIMA or SVR separately. T. Zhou, F. Wang and Z. Yang in [75] showed that SVM could be successfully combined with DWT preprocess. This hybrid model called WSVM provide the most precise nonstationary groundwater depth series predictions compared to ANN, SVM, and WA-ANN models. This paper reaffirms the efficiency of hybrid models in nonstationary time series forecasting.

A large number of existing studies in the broader literature have confirmed that machine learning methods usually achieve a quite high generalization performance in solving nonstationary and nonlinear time series forecasting problems and usually outperform traditional statistical time series prediction models. Obviously, each time series under consideration is unique and has specific properties that need to be taken into account when trying to predict its future values as accurately as possible. Therefore, a vast amount of papers has been published focusing on specific time series of interest to the authors and using different machine learning methods for time series forecasting in an attempt to find the best one for a particular case. Even when analyzing time series from the same domain, some studies show that some method is best suited for predicting such time series, while another research finds another best-suited model that outperforms the latter. Different findings of various researches are determined not only by different data used for forecasting but also by a different choice of data preprocessing method (if used), parameter selection technique, model architecture construction, the inclusion of additional improvements or other reasons. However, the large number of published researches allows noting the common main characteristics of different machine learning methods used for time series forecasting. NN and SVM based models are, probably, the most widely used, analyzed, and developed compared to other machine learning methods. G. Zhang, B. E. Patuwo and M. Y. Hu in [72] presented a state of the art discussion about the recent works in ANN for time series forecasting. A huge survey, based on publications and information found in other informative sources, of SVM applications for time series prediction is presented in [57]. All information in these two broad reviews and all literature sources used in this literature review are summarized by presenting the main general advantages and disadvantages of the NN and SVM based models in the Tables 1 and 2.

The literature review shows that there is an extended amount of works concerned on stationary and nonstationary time series forecasting while the research in the nearly nonstationary time series forecasting remains quite limited. Before describing the nearly nonstationary time series, it could be noted that one of the most commonly used ways for the testing of the stationarity in real world time series is to check the presence or absence of unit roots in an appropriate, for example autoregressive, representation of a given time series. It is well known that when the autoregressive characteristic equation of the process/time series has no unit roots, the process is assumed as stationary and therefore exhibits mean reversion in that it moves towards the long-term mean. Alternatively, if the process has a unit root, its trajectory is very fluctuant/unstable without the clear tendency of tending to any particular point that indicates the absence of having a mean reverting property. Such processes are characterized as nonstationary and, it is said that they follow a random walk. [38] Therefore, based on the unit root presence it seems that the processes could be divided into two groups: stationary and nonstationary. Even the unit root tests, such as Dickey-Fuller (DF), which is one of the best known and most widely used, Augmented Dickey-Fuller (ADF), Phillips-Perron (PP), Ng and Perron (NGP) or others, are used to confirm the null hypothesis that the process/time series is nonstationary and if it is denied the alternative hypothesis that the process is stationary is accepted [24]. However, it was noticed that the stationary processes, which has a root very close

ADVANTAGES	**DISADVANTAGES**
ANN	
1. Data-driven self-adaptive methods well suited for problems whose solutions require knowledge that is difficult to specify but for which there are enough data or observations. 2. Nonlinear approaches which are capable of performing nonlinear modeling without a prior knowledge about the relationships between input and output variables. 3. Can generalize and often correctly infer the unseen part of data even if the sample data contain noisy information. 4. Universal functional approximators which are able to approximate any continuous function to any desired accuracy. 5. Not dependent on linear, stationary processes.	1.Accuracy of predictions strongly depends on the architecture of network selection. 2. Have a large number of free parameters to be estimated. 3. Are prone to have overfitting and overtraining problems. 4. There are no structured methods to find the optimal architecture of ANN, therefore, time consuming experiments and trial-and-error procedures are often used. 5. Black-box methods because there is no explicit form to explain and analyze the relationship between inputs and outputs. 6. For static linear processes with little disturbance, they may not be better than linear statistical methods. 7. Not guaranteed to converge to optimal solution. 8. Could not handle both linear and nonlinear patterns equally well. 9. Usually require more data and computer time for training.
RBFN (compared to ANN)	
1. Have simpler fixed three-layer architecture and often are easier to train. 2. Are locally tuned.	1. The choice of basis function is problem dependent. 2. Accuracy of predictions strongly depends on the number of RBF units and parameters of RBF units.
RNN (compared to ANN)	
1. Can learn the temporal dependence from the data and use historical information for making further predictions that usually helps to achieve higher accuracy in forecasting.	1. The problem of vanishing gradient or exploding gradient is very common (could be solved using LSTM). 2. Are hard to train. 3. Are not able to keep track of long-term dependencies (could be solved using LSTM).

Table 1. The advantages and disadvantages of neural network models.

ADVANTAGES	**DISADVANTAGES**
SVR	
1. The quality and complexity of the solution can be independently controlled, irrespective of the dimension of the input space. 2. Based on the SRM principle they are typically superior compared to ANN in their ability to generalize. 3. Guaranteed to converge to optimal solution. 4. Nonlinear aspect of the prediction problem. 5. Not dependent on linear, stationary processes. 6. Has a small number of free parameters. 7. Not model dependent. 8. The sparse representation of solution.	1. Accuracy of predictions strongly depends on the parameters and kernel function selection. 2. There is no optimal method for the adaptation of free parameters as well as no formal proof of optimality for the selection of kernel function. 3. Cannot accurately forecast the linear basic part of time series. 4. Usually require long training time and enormous amount of computation when the data set, used for training, is large.
LS-SVR (compared to SVR)	
1. The transformation of QPP into a simultaneous linear system problem simplifies the computations. 2. Are preferred for large scale regression problems due to a high efficiency solution procedure.	1. The sparseness of the solution is lost.
DSVR (compared to SVR)	
1. Can deal with structural changes in data due to the different weighting of data points. 2. Are more robust in nonstationary time series forecasting. 3. Could result in a sparser representation of the solution.	1. The selection of the weight function and the procedure used to find optimal control rates needs to be more investigated.

Table 2. The advantages and disadvantages of models based on SVR.

to unity, usually act more likely to nonstationary processes and this led to the introduction of an additional group of processes called as nearly nonstationary. Such processes are still of great interest and require further investigation, especially in the problems of forecasting.

It could be found in the related literature that the unit root tests usually have low power against stable processes with roots near unity [17, 24, 49]. In other words, they usually approve the null hypothesis when the nearly nonstationary processes are tested, that in forecasting problems suggest, for example, to apply difference for making the process stationary, which theoretically is already assumed as stationary. When the difference is applied for the nearly nonstationary process, it is noninvertible and such process is called overdifferenced [56]. These findings about the inability of unit root tests to deny the null hypothesis which, based on their main idea, should be denied, have caused concern in econometrics and it was started to consider how it could affect the forecasting of processes. It is known that before applying models for the time series forecasting it is very important to check the stationarity of the time series which helps to choose an appropriate model and improve our predictions, therefore, for example, overdifferencing of the process can cause high inefficiency in forecasting [17]. The idea of using unit root tests in forecasting problems was to avoid overdifferencing and clearly separate stationary and nonstationary processes, therefore their inability to distinguish in a finite sample the null hypothesis from the alternative one, when the unit root of the process is close to unity, attracted attention of researchers.

Authors of [17] conducted a Monte Carlo study to verify the usefulness of unit root tests for selecting forecasting models and explored the extent to which pretesting for unit roots improves the accuracy of predictions in a canonical first order autoregressive (AR(1)) model with the trend, for a variety of sample sizes, forecast horizons, and degrees of persistence, corresponding to different autoregressive parameter values. They compared the performance of three forecasting models: AR(1) in levels with the linear deterministic trend by naming it as L, random walk with drift (D) and the model suggested by DF unit root pretest using 5% finite-sample critical values (P) (if the null hypothesis is approved then the model D is used, alternatively - the model L is applied). The performance of models was evaluated by its unconditional prediction mean squared error (PMSE) (see more in [12]) in 20000 Monte Carlo trials and for each value of autoregressive parameter/degree of persistence (for the simplicity it could be defined as β), equal to 1, 0.99, 0.97, 0.9 or 0.5, the ratios PMSE(D)/PMSE(L), PMSE(D)/PMSE(P) and PMSE(P)/PMSE(L) were calculated for all selected combinations of sample sizes (minimum value is 25, maximum - 1000) and forecast horizons (values ranging from 1 to 100). Based on the PMSE ratios it was confirmed that for the clearly nonstationary processes ($\beta = 1$) the DF pretest is unlikely to reject the model in differences and D model, which is actually the true model of such processes, is uniformly more accurate than the L model. Accordingly, for the obviously stationary processes ($\beta = 0.5$) P and L almost always tend to coincide and these models uniformly dominate the D model by showing that falsely adopting the model in differences can cause a very high loss in forecasting accuracy. These results are not unexpected and more interesting are those obtained by forecasting the processes with the value of β closer to 1. It was found that for $\beta = 0.99$ the pretest is unlikely to reject the null hypothesis and assumes theses processes as nonstationary, while for β equal to 0.97 or 0.9 the DF test lacks power and rarely rejects the null hypothesis only for small or moderate sample size and by growing of sample size the unit root null is rejected more often. The results of the research show that neither D nor L dominates always when the processes with β equals to 0.99, 0.97 or 0.9 are predicted and forecast accuracy depends on β, sample size and forecast horizon. It is interesting that for all of these three β values predictions from L are more accurate than those from D when the sample size is bigger while by taking the smaller samples the loss in forecast accuracy from estimates of L is much greater than the loss from inappropriately using the model in difference, especially for long forecast horizons. According to the authors, the poor relative performance of the L model for small sample size and large forecast horizons is due to the magnified distortions resulting from the DF small-sample bias, which plague the L model, as forecast horizon grows. Also, using L for processes with large unit roots in small samples, especially at long forecast horizons, there is an occasional possibility of L to draw some explosive forecasts and these predictions with extremely large errors dominate the PMSE which becomes much bigger compared to the one that is obtained using D, despite the fact that the PMSE of D worsens for longer forecast horizons. Besides, authors apply the simulation with a parameterization which is likely to be representative for some real world macroeconomic data and the results obtained approve the fact that for the processes with the unit root very close to unity, such as 0.99 or sometimes even 0.97, unit root tests are tend to approve the null hypothesis and it is not recommended to use always the L model, intended tor the stationary time series forecasting. By summing up all results, authors conclude that ”the best forecasting model is not necessarily the true model” and the inability of unit root tests to select the true model for the processes having a root near unity could be very helpful in the forecasting problems. This quite wide review of the research in [17] shows that the forecasting of the nearly nonstationary time series is a quite difficult problem and the accuracy of forecasting of such processes could be very dependent on the factors such as sample size and forecast horizon, however, the similarities between nearly nonstationary and nonstationary processes definitely exist in forecasting problems. In the research in [8], similarly as in this work, one-step ahead forecasting for the samples (of length 100) generated from the AR(1) process without a trend component (differently from the article [17]) is performed using an autoregressive model in levels and an autoregressive model in differences. The results of this research show that for the generated processes with $1 \geqslant \beta \geqslant 0.9$ the model in differences is superior and based on the Said-Dickey and Philips-Perron unit root tests it could be also confirmed that for such processes the null hypothesis tends to be approved. Authors of [49] analyze the predictions of the nearly nonstationary AR(1) processes with additive outliers by using the standard multistep-ahead predictor for a Gaussian AR(1) process and two unit root tests (DF and SSL, proposed by Shin et al. (1996)). The results of this research also confirm that the unit root tests can improve the accuracy of forecasts when β is very close to 1 (but smaller than 1) even if there are outliers in the process, at least for the three-step ahead predictions and quite small samples. J. H. Stock in [64] analyzed the forecasting of the univariate AR(1) without trend and with a root local to unity by using ordinary least squares (OLS) levels and random walk models with 2 unit root tests (DF and DF-GLS proposed by Elliot et al. (1996)). Final conclusions in [64] are, in principle, similar to those in [17] but it could be noted that Stock additionally applied models for the AR(1) simulations with β bigger than 1 (1.01 and 1.02). Such processes with $|\beta| > 1$ are explosive and the variance of them grows exponentially with a time [16], also they are rarely met in practice and usually not analyzed. Nevertheless, it could be noticed in the research that unit root tests approve the null hypothesis for such processes but the forecast errors obtained are larger, especially when $\beta = 1.02$, compared to those which are given for the processes with, for example, $1 \geqslant \beta \geqslant 0.97$. Also, it is interesting that when $\beta = 1.01$ the lower forecast error is given by using random walk model, while when $\beta = 1.02$ OLS levels model performs better, at least when the sample size is quite small (100 observations) and forecast horizon is 2, 10, 20 or 50.

Based on all the literature related to the nearly nonstationary time series forecasting which have been discussed here, it could be noticed that there obviously exist some similarities between nonstationary and nearly nonstationary time series. This conclusion could be made due to the fact that for the nearly nonstationary time series unit root tests are often tend to approve the null hypothesis that proposes for us to use forecasting models, intended for the nonstationary time series forecasting, and thus usually helps to reach a better forecasting accuracy. The literature review shows that ML algorithms could be very useful for the nonstationary time series forecasting and sometimes even outperform traditional statistical forecasting models, however all attempts to find the relevant studies considering specifically nearly nonstationary time series forecasting using ML models were unsuccessful. If we assume that models which are suitable for the nonstationary time series forecasting are appropriate for the nearly nonstationary time series forecasting then it could be said that we definitely could use ML models to make the predictions of the nearly nonstationary time series.

However, some questions remain unanswered, such as if the ML models could outperform traditional statistical models in the nearly nonstationary time series forecasting, if the same architecture of the ML model which is selected as optimal for the nonstationary time series is also optimal for the nearly nonstationary time series or which ML model helps to obtain highest forecast accuracy for the nearly nonstationary time series. These questions are an area of interest in this work.

2 Preliminarities

This section introduces the main definitions and concepts of the first order autoregressive processes, nonstationary and nearly nonstationary time series, forecasting models, which have been chosen in the experiment of this work and forecasting performance metrics.

2.1 First order autoregressive process (AR(1))

Autoregressive processes (AR) are very important in applications of statistics and time series forecasting. The idea of the autoregressive models is to explain the present value of the time series x_k by a function of p past values, $x_{k-1}, x_{k-2}, \ldots, x_{k-p}$. The value of p determines the number of time periods into the past, used for making a prediction of the current value of the time series, and it is also called the order of the autoregressive processes AR(p). [68] In this work the main focus is given on the first order autoregressive process AR(1) which could be very useful in analyzing the time series forecasting models due to the fact that its structure is simple and interpretable in a wide range of contexts. The AR(1) is defined by the equation:

$$x_k = \beta x_{k-1} + \epsilon_k, \quad k \in 1, 2, \ldots, \quad x_0 = 0, \tag{1}$$

where (ϵ_k) are random disturbances (innovations or error terms) at time k which usually are independent and identically distributed (i.i.d.) as $\mathcal{N}(0, \sigma^2)$ and β is an unknown parameter to be estimated [20, 45]. Parameter β is very important in the autoregression, for example, when $|\beta| < 1$, the AR(1) process, defined in the Eq. 1, is said to be stable and over time it becomes less affected on the changes in the past, while in the case $|\beta| > 1$, the process is explosive and it means that the effect of the changes in the past increases with time [45]. As it was noted in the literature review in the Section 1, the explosive processes are quite rarely applied for the practical purposes because usually time series which are less affected on the past are analyzed, therefore typically values of the AR(1) coefficient β are assumed to be less than one (or equal to one). Also, it is well known from the literature review that when $|\beta| < 1$ in the Eq. (1), the AR(1) process is stationary while when $|\beta| = 1$, the process is nonstationary. Usually, nonstationary time series with $\beta = 1$ is called random walk [16].

In practice, by creating an autoregressive representation of a real world time series under consideration, it is required to estimate the unknown parameter β. For this purpose such methods as Yule-Walker equations (method of moments) or maximum likelihood estimate (MLE) are used but customarily, the least squares estimator (LSE) is applied [20, 45]:

$$\widehat{\beta} = \frac{\sum_{k=1}^{n} x_{k-1} x_k}{\sum_{k=1}^{n} x_{k-1}^2}.$$

Based on the related literature (see, for example, [20, 45]) some important properties of the standartized LSE of β could be shortly presented. One of them is that for $|\beta| < 1$ the standartized LSE of β is asymptotically normal if ϵ_t are i.i.d.:

$$\left(\sum_{k=1}^{n} x_{k-1}^2 \right)^{1/2} (\widehat{\beta} - \beta) \xrightarrow[n\to\infty]{\mathbb{R}} \mathcal{N}(0, 1), \tag{2}$$

where $\xrightarrow[n\to\infty]{\mathbb{R}}$ is a notation of the convergence in distribution $\mathbb{R}$. However, if $|\beta| = 1$ the limiting law of the standardized LSE of β is non-normal and it has been showed that:

$$\left(\sum_{k=1}^{n} x_{k-1}^2 \right)^{1/2} (\widehat{\beta} - 1) \xrightarrow[n\to\infty]{\mathbb{R}} \frac{\frac{1}{2}(W^2(1) - 1)}{\left(\int_0^1 W^2(t)dt \right)^{1/2}}, \tag{3}$$

where W denotes a standard Wiener process $(W(t), t \in [0,1])$.

Also, in this work a great attention is paid on the nearly nonstationary time series and, based on the literature review, the AR(1) process is assumed to be nearly nonstationary when the parameter β is close to one or, in other words, the process has a root near unity. The nearly nonstationary AR(1) is generated as a triangular array [45]:

$$x_{n,k} = \beta_n x_{n,k-1} + \epsilon_k, \quad k = 0,1,\dots,n, \quad n = 1,2,\dots, \tag{4}$$

where $\beta_n \to 1$, as $n \to \infty$. Most commonly, β_n in the Eq. (4) is replaced by using the parameterization $\beta_n = e^{\gamma/n}$, where γ is a some fixed real number, or $\beta_n = 1 - \gamma/n$ with $\gamma > 0$ [20, 45]. Therefore, based, for example, on the first parameterization and AR(1) model, defined in the Eq. (4), it could be noticed that:

- when $\gamma = 0$, then $\beta_n = 1$ and process is nonstationary;
- when $\gamma < 0$ and n is fixed, then $0 < \beta_n < 1$ and process is stationary;
- when $\gamma > 0$ and n is fixed, then $\beta_n > 1$ and process is explosive (blows up in finite sample);
- when $\gamma < 0$ and γ/n is close to 0, then β_n is close to 1 and process is nearly nonstationary.

Some other parameterizations of β_n could be found in [45].

The literature review in the Section 1 shows that by using unit root tests for the AR(1) processes with a root near unity, despite the fact that theoretically they are assumed to be stationary, the null hypothesis, that the process is nonstationary, is often approved. Therefore, it was started to doubt if the approximation (2) is satisfactory for the nearly nonstationary processes. G. B. A. Evans and N. E. Savin (1981, 1984) in [21, 22] showed that the statistical properties of the LSE of β and associated unit root tests in a stationary AR(1), having a root near unity, are very similar to those which has a random walk. Based on that, Evans and Savin asserted that for the nearly nonstationary processes the approximation (3) can be used. However, later researches showed that neither (2) nor (3) are appropriate to approximate the distribution of standardized LSE of β for the AR(1) processes with a root near unity. [45] Many investigations in the literature have been proposed for the limit distribution of the standardized LSE of the coefficient in the nearly nonstationary AR(1) and some of them are summarized in [45]. For example, P. C. B. Phillips (1987) in [51] showed that:

$$n(\widehat{\beta}_n - \beta_n) \xrightarrow[n\to\infty]{\mathbb{R}} \frac{\int_0^1 U_\gamma(t)dW(t) + \frac{1}{2}\left(1 - \frac{\sigma}{\sigma'}\right)}{\int_0^1 U_\gamma^2(t)dt}, \tag{5}$$

where $\gamma < 0$, $\sigma' = lim_{n\to\infty}\mathbb{E}\left(n^{-1}(\sum_{k=1}^{n}\epsilon_k)^2\right)$ and U_γ is an Ornstein-Uhlenbeck process $(U_\gamma(t), t \in [0,1])$. The approximation (5) is used when innovations ϵ_k are strong mixing while when they are i.i.d. we have [45, 51]:

$$n(\widehat{\beta}_n - \beta_n) \xrightarrow[n\to\infty]{\mathbb{R}} \frac{\int_0^1 U_\gamma(t)dW(t)}{\int_0^1 U_\gamma^2(t)dt}.$$

2.2 Autoregressive Integrated Moving Average (ARIMA) model

Autoregressive Integrated Moving Average (ARIMA) model is one of the most commonly used models for the time series forecasting [48]. In the acronym ARIMA the combinations of letters "AR", "I", "MA" stand for autoregressive, integrated and moving average accordingly. In other words, ARIMA uses the autoregressive (AR(p)) and moving average (MA(q)) models with d-th order differencing of time series. Assume that x_t denotes the observation of time series at time t,

ϵ_t is an i.i.d. random error term at time t with zero mean and time is a discrete variable. The AR(p) model is a linear combination of p past observations of time series and could be written as [40, 48]:

$$x_t = \sum_{i=1}^{p} \alpha_i x_{t-i} + \epsilon_t, \tag{6}$$

where α_i are unknown parameters.

The MA(q) model is a linear combination of q past forecast errors (also called random shocks or innovations) and mathematical expression of this model is [40, 48]:

$$x_t = \sum_{i=1}^{q} \beta_i \epsilon_{t-i} + \epsilon_t, \tag{7}$$

where β_i are unknown parameters.

As it was mentioned in the literature review in the Section 1, differencing is a commonly used method which helps to make the series stationary by taking d differences of nonstationary time series. For example, the first order differences of x_t are computed by $\nabla x_t = x_t - x_{t-1}$, then the second order differences are given by $\nabla^2 x_t = \nabla x_t - \nabla x_{t-1}$, therefore the d-th order differences of x_t are computed by [40]:

$$\nabla^d x_t = \nabla^{d-1} x_t - \nabla^{d-1} x_{t-1} \tag{8}$$

By making a combination of the expressions in Eq. (6), (7) and (8) the ARIMA(p,d,q) model could be written as [40]:

$$\nabla^d x_t = \sum_{i=1}^{p} \alpha_i \nabla^d x_{t-i} + \sum_{i=1}^{q} \beta_i \epsilon_{t-i} + \epsilon_t, \quad \alpha \in \mathbb{R}^p, \quad \beta \in \mathbb{R}^q.$$

In addition, it could be noted that when the stationary time series is forecasting and the differencing is not used, then ARIMA(p,0,q) and ARMA(p,q) models are equivalent (just a combination of AR(p) and MA(q)) and could be written as:

$$x_t = \sum_{i=1}^{p} \alpha_i x_{t-i} + \sum_{i=1}^{q} \beta_i \epsilon_{t-i} + \epsilon_t.$$

Therefore, it could be noticed that,for example, for the stationary AR(1) time series (defined in the Eq. (1)) the ARIMA(1,0,0) is the equivalent/true model and the prediction of the observation at the moment t is given by:

$$\widehat{x}_t = \alpha_1 x_{t-1} + \epsilon_t.$$

Accordingly, for the nonstationary AR(1) time series or random walk the ARIMA(0,1,0) is the equivalent/true model, thus:

$$\widehat{x}_t = x_{t-1} + \epsilon_t.$$

2.3 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) are one of the main tools used in the machine learning. Based on the literature review, the simplest and usually used ANNs are feedforward networks in which the connections between nodes do not form a cycle and the information is only processed in one direction, from the input to the output. One of the most important feedforward ANNs are Multilayer perceptrons (MLPs).

2.3.1 Multilayer Perceptron (MLP)

Multilayer perceptrons (MLPs) are, probably, the most widely used ANNs in the forecasting problems. MLP networks are constructed of multiple layers of nodes/neurons and each neuron in one layer is directly connected to the neurons of the subsequent layer. MLP has to be composed of a minimum of three layers and consists of input layer, one or more hidden layers and output layer. In addition, bias neurons, connected to each unit in the hidden and output layers, could be included. The bias neuron has a value of positive one and is analogous to the intercept term in a regression equation. Figure 1 illustrates the diagrammatically depiction of the three-layer MLP for the one-step forecasting.

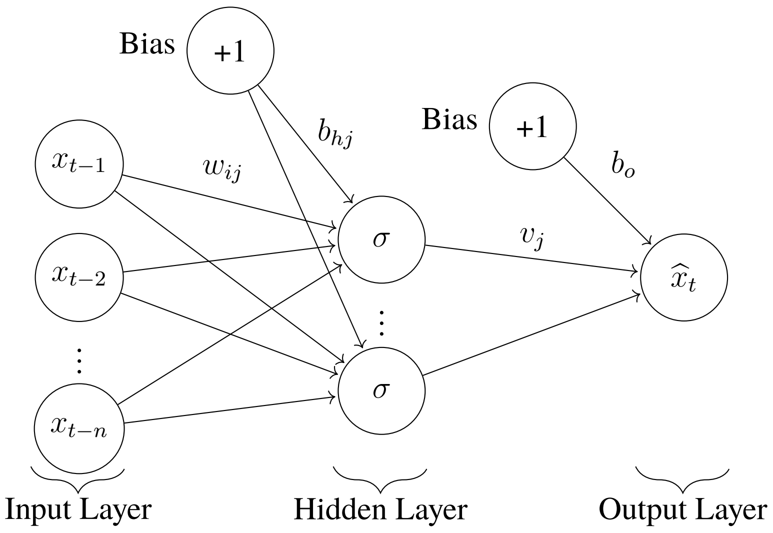


Figure 1. The architecture of the three-layer MLP (created by the author).

Input layer consists of n nodes representing the input features $x_{t-1}, x_{t-2}, \ldots, x_{t-n}$ used to find the value of prediction of the time series value x at the moment t ($\widehat{x}_t$) which is represented by a single output node in the output layer. Hidden layer contains m nodes and each j-th ($j = 1, 2, \ldots, m$) hidden neuron is connected with each i-th ($i = 1, 2, \ldots, n$) input node by the weighted connection with the coefficient w_{ij} for weighting. Every hidden node transforms the input values with a linear summation of weighted inputs by adding bias terms b_{hj}, followed by a nonlinear activation function σ. There are many different functions that can be used as the nonlinear activation function, such as rectified linear unit (ReLU) or hyperbolic tangent (tanh) (see Table 3), but usually the logistic sigmoid function is applied [3]:

$$\sigma(u) = \frac{1}{1 + e^{-u}}.$$

The output values of activation functions are multiplied by weights v_j then summed by adding the bias term b_o and the obtained value is equal to $\widehat{x}_t$. The computation of the output value could be written using the following mathematical expression [3, 5]:

$$\widehat{x}_t = \sum_{j=1}^{m} v_j \sigma \left(\sum_{i=1}^{n} w_{ij} x_{t-i} + b_{hj} \right) + b_o, \quad \forall t.$$

Before using MLP for the time series forecasting the weights and biases must be initialized. Usually, they are set to random numbers. The main idea of the training neural network is to find optimal weights and biases that minimize prediction error and for this purpose MLP commonly uses the backpropagation algorithm. [3, 30, 67]

2.3.2 Backpropagation algorithm (BP)

Backpropagation algorithm is, probably, the most common training procedure for the MLP network. It employs an iterative optimization method, known as gradient descent, to compute and update the connection weights and biases in an attempt to minimize the error between the output of an MLP network and the desired output. After each forward pass through the network, backpropagation algorithm is used to perform a backward pass by calculating the gradient of the error function with respect to the networks parameters (weights and biases) [5].

For the simpler notation, each hidden node and a single output node of the three-layer MLP, illustrated in Figure 1, are denoted as h_j and o respectively. The input and the output of the j-th hidden node could be denoted as z_{hj} and a_{hj} respectively:

$$z_{hj} = \sum_{i=1}^{n} w_{ij} x_{t-i} + b_{hj}, \tag{9}$$

$$a_{hj} = \sigma(z_{hj}).$$

The output node also has its input and output that could be denoted as z_o and a_o respectively:

$$z_o = \sum_{j=1}^{m} v_j a_{hj} + b_o, \tag{10}$$

$$a_o = \phi(z_o). \tag{11}$$

Here ϕ is the activation function applied to the output node. Usually, the linear/identity function is used in the output layer [67], thus, the value of this function is equal to the input value of the function (see Table 3) and also it could be noticed that the output of the node o is equal to the predicted value of x_t:

$$a_o = z_o = \widehat{x}_t.$$

To evaluate how close $\widehat{x}_t$ is to the expected output x_t the error/loss function E is used. One of the most common choices is the summed squared error (SSE) function [5] and based on the considered one-step forecasting problem it could be written as follows:

$$E = \frac{1}{2}(x_t - \widehat{x}_t)^2 = \frac{1}{2}(x_t - a_o)^2. \tag{12}$$

The $1/2$ is added to ease the calculation of derivative of the loss function.

The relation between the whichever input node, j-th hidden node and the output node of the MLP network could be diagrammatically illustrated as below by including denoted inputs and outputs of the hidden h_j and output o nodes: The diagram above (Figure 2) shows the forward pass

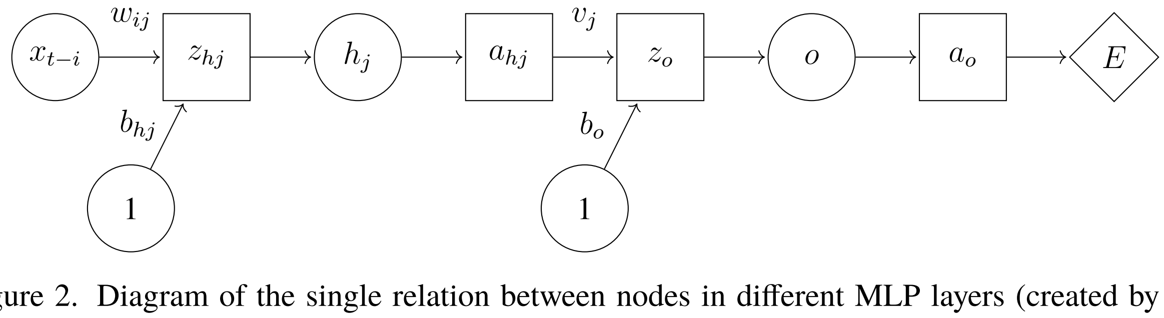


Figure 2. Diagram of the single relation between nodes in different MLP layers (created by the author).

through the network and the main idea of backpropagation is to go backwards by calculating the partial derivatives of the loss function with respect to the each weight and bias of the network. All gradients are computed using the chain rule [5] due to the fact that the loss function does not directly depend on network parameters. By going backwards, first of all, the derivatives of the loss function with respect to the weights between hidden and output layers are calculated [5, 7]:

$$\frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial a_o}\frac{\partial a_o}{\partial v_j} = \frac{\partial E}{\partial a_o}\frac{\partial a_o}{\partial z_o}\frac{\partial z_o}{\partial v_j}. \tag{13}$$

Based on Eq. (12):

$$\frac{\partial E}{\partial a_o} = \frac{\partial}{\partial a_o}\left(\frac{1}{2}(x_t - a_o)^2\right) = (x_t - a_o)(-1) = -(x_t - a_o) = -(x_t - \widehat{x}_t). \tag{14}$$

The second of the three derivatives in the last part of the Eq. (13) is equal to the 1 due to the fact that the a_o is the linear/identity function giving the constant value. By taking the partial derivative of z_o with respect to v_j and using Eq. (10) gives:

$$\frac{\partial z_o}{\partial v_j} = \frac{\partial}{\partial v_j}\left(\sum_{j=1}^{m} v_j a_{hj} + b_o\right) = a_{hj}, \tag{15}$$

since only one of the terms in the sum is related to the specific weight v_j.

By substituting Eq. (14) and (15) into Eq. (13) we have the derivative of E with respect to v_j as:

$$\frac{\partial E}{\partial v_j} = -(x_t - \widehat{x}_t)a_{hj}. \tag{16}$$

Assume that $-(x_t - \widehat{x}_t) = \delta_o$ and the above relation in Eq. (16) can be rewritten as:

$$\frac{\partial E}{\partial v_j} = \delta_o a_{hj}. \tag{17}$$

The negative gradient of the loss function defined in Eq. (17) multiplied by the constant called the learning rate or step size (η), which determines the gradient's influence, shows the level of adjustment (Δv_j) of the weight v_j and adjusted weight could be defined as v_j^* [5]:

$$v_j^* = v_j + \Delta v_j = v_j - \eta\frac{\partial E}{\partial v_j} = v_j - \eta\delta_o a_{hj}.$$

In principle, the partial derivatives of the loss function with respect to the weights between input and hidden layers are calculated similarly [5, 7]:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_{hj}}\frac{\partial a_{hj}}{\partial w_{ij}} = \frac{\partial E}{\partial a_{hj}}\frac{\partial a_{hj}}{\partial z_{hj}}\frac{\partial z_{hj}}{\partial w_{ij}}. \tag{18}$$

Taking all dependencies into account and using chain rule the partial derivative of E with respect to a_{hj} gives:

$$\frac{\partial E}{\partial a_{hj}} = \frac{\partial E}{\partial z_o}\frac{\partial z_o}{\partial a_{hj}} = \frac{\partial E}{\partial a_o}\frac{\partial a_o}{\partial z_o}\frac{\partial z_o}{\partial a_{hj}}. \tag{19}$$

From Eq. (13) and (17):

$$\frac{\partial E}{\partial a_o}\frac{\partial a_o}{\partial z_o} = \delta_o \tag{20}$$

The partial derivative of z_o with respect to a_{hj} is calculated based on Eq. (10):

$$\frac{\partial z_o}{\partial a_{hj}} = \frac{\partial}{\partial a_{hj}}\left(\sum_{j=1}^{m} v_j a_{hj} + b_o\right) = v_j. \tag{21}$$

Substituting Eq. (20) and (21) in Eq. (19) we obtain:

$$\frac{\partial E}{\partial a_{hj}} = \delta_o v_j. \tag{22}$$

It is known that the logistic sigmoid function is applied in the hidden layer, thus the derivative of the output of the j-th hidden node a_{hj} with respect to its input z_{hj} is given by (see Table 3):

$$\frac{\partial a_{hj}}{\partial z_{hj}} = a_{hj}(1 - a_{hj}). \tag{23}$$

The last partial derivative in Eq. (18) is calculated based on Eq. (9):

$$\frac{\partial z_{hj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{i=1}^{n} w_{ij} x_{t-i} + b_{hj} \right) = x_{t-i}. \tag{24}$$

Substituting Eq. (22), (23) and (24) in (18) gives:

$$\frac{\partial E}{\partial w_{ij}} = \delta_o v_j (a_{hj}(1 - a_{hj})) x_{t-i}. \tag{25}$$

The part of the Eq. (25), which is able to change during the training while network parameters are adjusting, is denoted as δ_{hj} and the partial derivative of the loss function with respect to each weight w_{ij} could be rewritten as:

$$\frac{\partial E}{\partial w_{ij}} = \delta_{hj} x_{t-i}.$$

Each weight w_{ij} is adjusted in the same way as v_j [5, 7]:

$$w_{ij}^* = w_{ij} + \Delta w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} = w_{ij} - \eta \delta_{hj} x_{t-i}.$$

Biases correspond to a weight of an additional edge with a fixed input of 1. Therefore, bias weights, for the simplicity called simply biases, are updated similarly as weights between nodes of different networks layers, except that there is no input from a previous layer in bias units. To find the partial derivative of the loss function with respect to b_o previous calculations in Eq. (13), (14) and (10) could be used:

$$\frac{\partial E}{\partial b_o} = \frac{\partial E}{\partial a_o}\frac{\partial a_o}{\partial b_o} = \frac{\partial E}{\partial a_o}\frac{\partial a_o}{\partial z_o}\frac{\partial z_o}{\partial b_o} = \delta_0 \frac{\partial}{\partial b_o} \left(\sum_{j=1}^{m} v_j a_{hj} + b_o \right) = \delta_0(1) = \delta_0.$$

Updated value of the b_o is given by [7]:

$$b_o^* = b_o + \Delta b_o = b_o - \eta \frac{\partial E}{\partial b_o} = b_o - \eta \delta_0.$$

The derivative of E with respect to b_{hj} is calculated based on Eq. (22), (23) and (9):

$$\begin{aligned}\frac{\partial E}{\partial b_{hj}} &= \frac{\partial E}{\partial a_{hj}}\frac{\partial a_{hj}}{\partial b_{hj}} = \frac{\partial E}{\partial a_{hj}}\frac{\partial a_{hj}}{\partial z_{hj}}\frac{\partial z_{hj}}{\partial b_{hj}} = \delta_0 v_j a_{hj}(1 - a_{hj}) \frac{\partial}{\partial b_{hj}} \left(\sum_{i=1}^{n} w_{ij} x_{t-i} + b_{hj} \right) \\ &= \delta_0 v_j a_{hj}(1 - a_{hj})(1) = \delta_0 v_j a_{hj}(1 - a_{hj}) = \delta_{hj}.\end{aligned}$$

Adjusted b_{hj} is written as [7]:

$$b_{hj}^* = b_{hj} + \Delta b_{hj} = b_{hj} - \eta \frac{\partial E}{\partial b_{hj}} = b_{hj} - \eta \delta_{hj}.$$

Using backpropagation as the training algorithm for the MLP network the number of backward passes is defined by the number of iterations/epochs and all parameters (v_j, w_{ij}, b_o, b_{hj}) are adjusted after each iteration. The updated weights and biases are used for training the network at each epoch until the initially predefined last iteration is achieved or the determined minimum error has been reached.

2.4 Recurrent Neural Networks (RNNs)

Fully connected Recurrent Neural Networks (RNNs) look quite similar to MLP networks because they also have input, hidden and output layers and use the forward pass technique to find predictions for the time series values. However, RNN is able to keep in memory previous computations in hidden units by saving them into so-called context units and later using this information of context units as inputs [39], while in MLP network inputs of hidden nodes are not dependent to previous outputs in separate test cases.

Suppose, that the architecture of RNN consists of three layers with n nodes in the input layer, m neurons in the hidden layer, a single node in the output layer and two additional bias neurons, like in MLP, illustrated in Figure 1. RNN with a single output node is used for the one-step forecasting and firstly we are trying to find the prediction of the time series value x at the moment t ($\widehat{x}_t$) based on the previous observations $x_{t-1}, x_{t-2}, \ldots, x_{t-n}$. After that, RNN could be applied again to find the following x prediction at the moment $t+1$ ($\widehat{x}_{t+1}$) and then the difference between MLP and RNN could be clearly seen. In this case MLP just takes $x_t, x_{t-1}, \ldots, x_{t-n-1}$ as input features, and the weighted sum of them by adding bias term is the input of each neuron in the hidden layer, while in RNN hidden nodes depend not only on input features but also rely on the output values of hidden nodes in the previous calculation when x value at the moment t was predicting. For the mathematical expressions the same notations could be used as in Section 2.3, by adding additional index $\tau = (1, 2, \ldots, T)$ which defines the time step or the stage of computations. Then the input and the output of the j-th hidden node at the time step τ could be written as:

$$z_{h\tau j} = \sum_{i=1}^{n} w_{ij} x_{\tau(t-i)} + \sum_{p=1}^{m} u_{pj} a_{h(\tau-1)pj} + b_{hj}, \tag{26}$$

$$a_{h\tau j} = \sigma(z_{h\tau j}). \tag{27}$$

Here $a_{h(\tau-1)pj}$ in the Eq. (26) defines the output of the p-th ($p = 1, 2, \ldots, m$) node in the hidden layer at the previous ($\tau - 1$) time step which is connected with each j-th hidden node at the current (τ) time step by the weighted connection with the coefficient u_{pj}. It could be noticed that at the first time step ($\tau = 1$) the network does not have any previous computations that need to be memorized and all m values of $a_{h(\tau-1)pj}$ are equal to 0, therefore the inputs of the hidden nodes in RNN and MLP have the same mathematical expressions. In each hidden node the nonlinear activation function σ is applied for the input $z_{h\tau j}$ and then the output of j-th hidden node is equal to the value of this function, similarly as in MLP. Equations of the most commonly used activation functions and their derivatives are presented in the Table 3.

Function	**Equation**	**Derivative**
Linear/Identity	$\sigma(u) = u$	$\sigma'(u) = 1$
Logistic sigmoid	$\sigma(u) = \frac{1}{1+e^{-u}}$	$\sigma'(u) = \sigma(u)(1 - \sigma(u))$
Hyperbolic tangent (tanh)	$\sigma(u) = \tanh(u) = \frac{2}{1+e^{-2u}} - 1$	$\sigma'(u) = 1 - \sigma(u)^2$
Rectified Linear Unit (ReLU)	$\sigma(u) = \begin{cases} 0 & \text{for } x \leqslant 0 \\ x & \text{for } x > 0 \end{cases}$	$\sigma'(u) = \begin{cases} 0 & \text{for } x \leqslant 0 \\ 1 & \text{for } x > 0 \end{cases}$

Table 3. Most commonly used activation functions [18, 74]

The input and output of the output node at the time step τ in RNN act exact the same as in MLP (see Eq. (10) and (11)) and are given by:

$$z_{o\tau} = \sum_{j=1}^{m} v_j a_{h\tau j} + b_o,$$

$$a_{o\tau} = \phi(z_{o\tau}).$$

From the above equations it could be noticed that all weights and biases are not dependent on the time step τ and that is because in RNN they are shared throughout the entire network and all time steps [39]. Therefore, the network does not change between time steps and only the inputs and outputs differ.

Suppose that we have vectors $\mathbf{x}_\tau = (x_{\tau(t-1)}, x_{\tau(t-2)}, \dots, x_{\tau(t-n)})$, $\mathbf{z}_{h\tau} = (z_{h\tau 1}, z_{h\tau 2}, \dots, z_{h\tau m})$, $\mathbf{a}_{h\tau} = (a_{h\tau 1}, a_{h\tau 2}, \dots, a_{h\tau m})$, $\mathbf{z}_{o\tau} = (z_{o\tau})$, $\mathbf{a}_{o\tau} = (a_{o\tau})$, $\mathbf{b}_h = (b_{h1}, b_{h2}, \dots, b_{hm})$ and $\mathbf{b}_o = (b_o)$ and matrices **W** of weights w_{ij}, **U** of weights u_{pj} and **V** of weights v_j where $i = (1, 2, \dots, n); \quad j, p = (1, 2, \dots, m)$. Using these notations Eq. (26), (27), (10) and (11) could be rewritten as [13, 39]:

$$\mathbf{z}_{h\tau} = \mathbf{W}\mathbf{x}_\tau + \mathbf{U}\mathbf{a}_{h(\tau-1)} + \mathbf{b}_h,$$

$$\mathbf{a}_{h\tau} = \sigma(\mathbf{z}_{h\tau}),$$

$$\mathbf{z}_{o\tau} = \mathbf{V}\mathbf{a}_{h\tau} + \mathbf{b}_o,$$

$$\mathbf{a}_{o\tau} = \phi(\mathbf{z}_{o\tau}).$$

A single RNN at the time step τ with one input I_τ, one hidden H_τ and one output O_τ layers is visualized on the left of the Figure 3. The extended RNN model in a time sequential manner is illustrated on the right of the Figure 3.

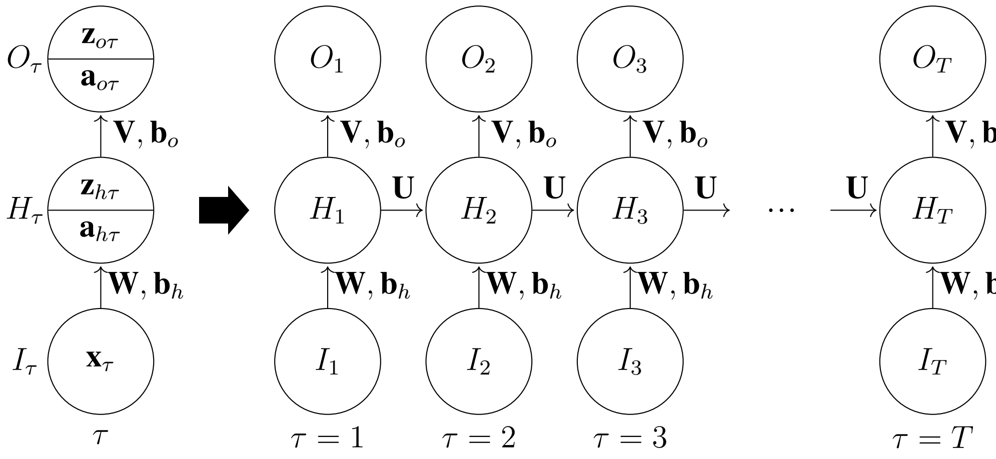


Figure 3. The architecture of the three-layer RNN (on the left) and RNN model unfolded in time (on the right) (created by the author).

Usually, RNN, similarly to MLP, uses backpropagation as its training algorithm. However, due to dependency between calculations in a current and previous time steps RNN requires a special form of backpropagation called Backpropagation Through Time (BPTT) [13].

2.4.1 Backpropagation Through Time algorithm (BPTT)

In principle, BPTT is very similar to standard BP which is widely presented in Section 2.3.2 and here are introduced only main concepts.

Suppose that E_τ is the loss function at the time step τ. For the purpose to introduce the main idea of BPTT it is better not focus on a specific function but there are many different functions that could be applied to calculate the loss between the output of algorithm and the given target value, such as most commonly used root mean squared error (RMSE), mean squared error (MSE), mean average error (MAE) or others (see more in the Section 2.6). The loss over all T time steps is obtained as follows:

$$E = \sum_{\tau=1}^{T} E_\tau.$$

Using BPTT, as well as standard BP, for the training of the network a backward pass through the different layers of the network are performed by calculating the gradient of the loss function with respect to networks weights and biases. BPTT consists of a repeated application of the chain rule. Due to the fact that all networks parameters are shared across the whole time sequence, when we are calculating the derivative of the total loss with respect to **V** it is possible to differentiate to it at each time step and sum all together [13]:

$$\frac{\partial E}{\partial \mathbf{V}} = \sum_{\tau=1}^{T} \frac{\partial E_\tau}{\partial \mathbf{V}} = \sum_{\tau=1}^{T} \frac{\partial E_\tau}{\partial \mathbf{a}_{o\tau}} \frac{\partial \mathbf{a}_{o\tau}}{\partial \mathbf{z}_{o\tau}} \frac{\partial \mathbf{z}_{o\tau}}{\partial \mathbf{V}}. \tag{28}$$

The derivative of the total loss with respect to $\mathbf{b}_o$ is given using Eq. (28) just replacing **V** with $\mathbf{b}_o$.

More difficult task is to find the derivative of E with respect to **U** which is obtained as follows [13]:

$$\frac{\partial E}{\partial \mathbf{U}} = \sum_{\tau=1}^{T} \frac{\partial E_\tau}{\partial \mathbf{U}} = \sum_{\tau=1}^{T} \frac{\partial E_\tau}{\partial \mathbf{a}_{o\tau}} \frac{\partial \mathbf{a}_{o\tau}}{\partial \mathbf{z}_{o\tau}} \frac{\partial \mathbf{z}_{o\tau}}{\partial \mathbf{a}_{h\tau}} \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{U}}. \tag{29}$$

Note that in RNN the sequential dependency between the outputs of the hidden layers at adjacent time steps exists, thus $\mathbf{a}_{h\tau}$ partially depends on $\mathbf{a}_{h(\tau-1)}$ which partially depends on $\mathbf{a}_{h(\tau-2)}$ and that is until we reach the last time step. Every hidden layer has a weighted connection with the hidden layer at the previous time step with the coefficient U for weighting. Consequently, the partial derivative of $\mathbf{a}_{h\tau}$ with respect to U could be calculated as [13]:

$$\frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{U}} = \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{U}} + \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{a}_{h(\tau-1)}} \frac{\partial \mathbf{a}_{h(\tau-1)}}{\partial \mathbf{U}} = \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{U}} + \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{a}_{h(\tau-1)}} \left(\frac{\partial \mathbf{a}_{h(\tau-1)}}{\partial \mathbf{U}} + \frac{\partial \mathbf{a}_{h(\tau-1)}}{\partial \mathbf{a}_{h(\tau-2)}} \frac{\partial \mathbf{a}_{h(\tau-2)}}{\partial \mathbf{U}} \right).$$

Calculation in the above equation continues using the same logic until we reach the last hidden layer at the time step $\tau = 1$. It could be shortened and written as:

$$\frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{U}} = \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{a}_{h\tau}} \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{U}} + \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{a}_{h(\tau-1)}} \frac{\partial \mathbf{a}_{h(\tau-1)}}{\partial \mathbf{U}} + \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{a}_{h(\tau-2)}} \frac{\partial \mathbf{a}_{h(\tau-2)}}{\partial \mathbf{U}} + \cdots + \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{a}_{h1}} \frac{\partial \mathbf{a}_{h1}}{\partial \mathbf{U}} = \sum_{k=1}^{\tau} \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{a}_{hk}} \frac{\partial \mathbf{a}_{hk}}{\partial \mathbf{U}}. \tag{30}$$

By substituting Eq. (30) in Eq. (29) we obtain:

$$\frac{\partial E}{\partial \mathbf{U}} = \sum_{\tau=1}^{T} \frac{\partial E_\tau}{\partial \mathbf{U}} = \sum_{\tau=1}^{T} \sum_{k=1}^{\tau} \frac{\partial E_\tau}{\partial \mathbf{a}_{o\tau}} \frac{\partial \mathbf{a}_{o\tau}}{\partial \mathbf{z}_{o\tau}} \frac{\partial \mathbf{z}_{o\tau}}{\partial \mathbf{a}_{h\tau}} \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{a}_{hk}} \frac{\partial \mathbf{a}_{hk}}{\partial \mathbf{U}}.$$

The derivative of E with respect to **W** is calculated analogously as the derivative of E with respect to **U** [13]:

$$\frac{\partial E}{\partial \mathbf{W}} = \sum_{\tau=1}^{T} \frac{\partial E_\tau}{\partial \mathbf{W}} = \sum_{\tau=1}^{T} \sum_{k=1}^{\tau} \frac{\partial E_\tau}{\partial \mathbf{a}_{o\tau}} \frac{\partial \mathbf{a}_{o\tau}}{\partial \mathbf{z}_{o\tau}} \frac{\partial \mathbf{z}_{o\tau}}{\partial \mathbf{a}_{h\tau}} \frac{\partial \mathbf{a}_{h\tau}}{\partial \mathbf{a}_{hk}} \frac{\partial \mathbf{a}_{hk}}{\partial \mathbf{W}}. \tag{31}$$

The derivative of the total loss with respect to $\mathbf{b}_h$ is given using Eq. (31) just replacing **W** with $\mathbf{b}_h$.

Further calculation of partial derivatives could be done based on equations presented in Section 2.3.2. After each iteration on which the BPTT is used to go backwards through the network all weights and biases are updated using the following equation:

$$A := A - \eta \frac{\partial E}{\partial A},$$

where A defines any of weights $(\mathbf{W}, \mathbf{U}, \mathbf{V})$ or biases $(\mathbf{b}_h, \mathbf{b}_o)$ and η is learning rate.

2.5 Support Vector Regression (SVR)

The Support Vector Regression (SVR) is an adaptation of the very powerful machine learning theory based tool, intended for the classification, namely Support Vector Machines (SVM), for the regression problems [71]. The concept of SVR could be illustrated by formulating a typical regression problem. Suppose that the set of data $T = \{(\mathbf{x}_i, y_i), i = 1, \dots, l\} \subset \mathcal{X} \times \mathbb{R}$ is considered, where $\mathcal{X}$ denotes the input space, $\mathbf{x}_i \in \mathbb{R}^N$ is an N-dimensional vector of the model inputs, $y_i \in \mathbb{R}$ is an actual value of a target, representing the desired output, and l is a total number of data patterns. One of the main aims in SVR is to determine a regression function $f(\mathbf{x})$ which could obtain values with a not larger than ϵ deviation from each of the actually obtained targets y_i of the considering data [62]. The regression problems could be classified as linear and nonlinear, thus, for the linear case the function $f(\mathbf{x})$ could be written as:

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b, \quad \text{with} \quad \mathbf{w} \in \mathcal{X}, b \in \mathbb{R}, \tag{32}$$

where $\mathbf{w}$ is a weight vector, $\mathbf{x}$ is the vector of input vectors $\mathbf{x_i}$, b is a constant (analogous to the intercept term in a regression equation) and $\langle \cdot, \cdot \rangle$ denotes the dot product in $\mathcal{X}$ [62].

The second basic aim in SVR is to ensure that $f(\mathbf{x})$ is as flat as possible and flatness in the case of Eq. (32) means a small $\mathbf{w}$ [14, 62]. That could be implemented by minimizing the norm of $\mathbf{w}$ ($\|\mathbf{w}\|$) which can be changed to $\frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle$ without changing the solution and ensuring mathematical convenience [52]. The regression problem could be written as a convex optimization problem [62]:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{subject to} \quad \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b \leqslant \epsilon \\ \langle \mathbf{w}, \mathbf{x}_i \rangle + b - y_i \leqslant \epsilon. \end{cases} \end{aligned} \tag{33}$$

The implicit assumption in the Eq. (33) exists that the convex optimization problem is feasible or in other words, that it is possible to find a function f that can approximate all pairs $(\mathbf{x}_i, y_i)$ with ϵ precision [62]. However, usually, due to the high variety of data in the time series, this optimization problem is not feasible or we just may want to allow for some errors, therefore, analogously to the modified version of SVM: the soft margin SVM, introduced by V. Vapnik and C. Cortes in 1995, the slack variables ξ_i, ξ_i^* could be included into the optimization problem (33), which help to cope with otherwise infeasible constraints [52, 62]. In other words, the slack variables ξ_i and ξ_i^* specify the upper and the lower errors respectively subject to an error tolerance ϵ. SVR uses the so-called Vapnik's ϵ-insensitive loss function, penalizing the errors between the predicted values and the desired outputs, which are bigger than the tolerated error ϵ [4]. Several types of loss functions exist, such as linear, quadratic or Huber, but the most commonly used is the linear ϵ-insensitive loss function which is defined as [4]:

$$L_\epsilon(y_i) = \begin{cases} 0 & \text{for} \quad |y_i - (\langle \mathbf{w}, \mathbf{x}_i \rangle + b)| \leqslant \epsilon \\ |y_i - (\langle \mathbf{w}, \mathbf{x}_i \rangle + b)| - \epsilon & \text{otherwise.} \end{cases}$$

The visual representation of the linear SVR and ϵ-insensitive loss function is illustrated in the Figure 4. In the graph on the left of the Figure 4 a straight line denotes the regression function $f(\mathbf{x})$ and the ϵ, usually called as the non-sensitive loss or the precision parameter [42], represents the radius (dashed lines) of the so-called tube located around the $f(\mathbf{x})$. The area between the dashed lines (in the tube) is called as ϵ-insensitive zone [42], where the loss function, represented in the graph on the right of the Figure 4, assumes zero value and the prediction errors with magnitudes equal to or smaller than ϵ are not penalized, while the pairs $(\mathbf{x}_i, y_i)$ outside the tube have the errors ξ_i or ξ_i^* which are penalized by the ϵ-insensitive loss function.

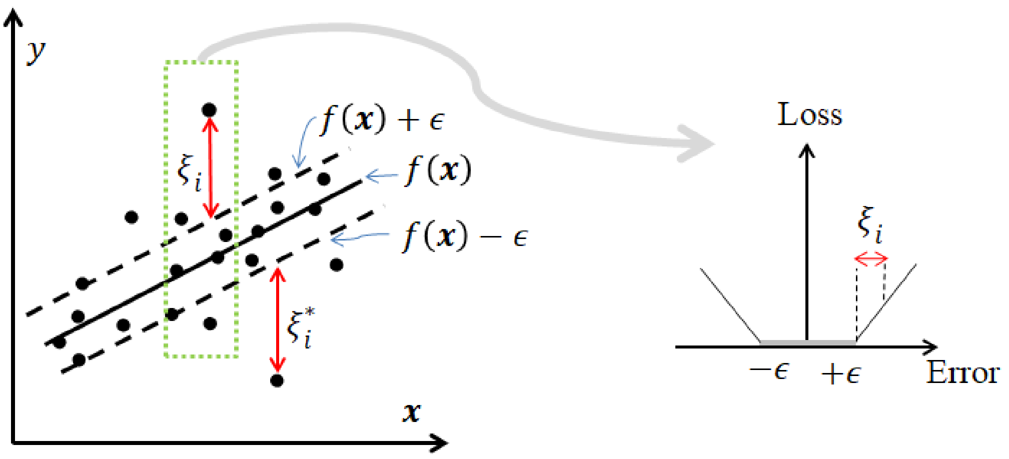


Figure 4. A schematic representation of the linear SVR (on the left) and ϵ-insensitive loss function (on the right) (created by the author).

After introducing the slack variables the convex optimization problem, defined in the Eq. (33), could be rewritten and has the following formulation:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i+\xi_i^*) \\ &\text{subject to} \quad \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x}_i\rangle - b \leqslant \epsilon + \xi_i \\ \langle \mathbf{w}, \mathbf{x}_i\rangle + b - y_i \leqslant \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geqslant 0, \quad i = 1, \dots, l, \end{cases} \end{aligned} \tag{34}$$

where the constant $C > 0$, usually called as the penalty parameter or the regularization constant, determines the trade-off between the flatness of f (small $\mathbf{w}$) and the degree to which deviations larger than ϵ are tolerated [62]. For example, larger value of C gives more weight to minimizing the error, while the smaller C gives more weight to minimizing the flatness [4].

By introducing Lagrange multipliers or dual variables $\alpha_i, \alpha_i^*, \eta_i,$ and η_i^*, which are nonnegative real numbers ($\alpha_i, \alpha_i^*, \eta_i, \eta_i^* \geqslant 0$), the Lagrange function, called Lagrangian, could be constructed from the objective function and the corresponding constraints (defined in the Eq. (34)) and written as [4, 62]:

$$\begin{aligned} L =& \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i+\xi_i^*) - \sum_{i=1}^{l}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ &- \sum_{i=1}^{l}\alpha_i(-y_i + \langle \mathbf{w}, \mathbf{x}_i\rangle + b + \epsilon + \xi_i) \\ &- \sum_{i=1}^{l}\alpha_i^*(y_i - \langle \mathbf{w}, \mathbf{x}_i\rangle - b + \epsilon + \xi_i^*) \end{aligned} \tag{35}$$

Lagrangian is optimized based on the Karush–Kuhn–Tucker (KKT) conditions. According to the KKT conditions, the partial derivatives of the Lagrangian with respect to the variables $b, \mathbf{w}, \xi_i$ and ξ_i^* are taken by setting them equal to zero while the partial derivatives of the Lagrangian with respect to the Lagrange multipliers return the constraints which are required to be less than or equal to zero [4, 62]:

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{l}(\alpha_i^* - \alpha_i) = 0 \tag{36}$$

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\mathbf{x}_i = 0 \tag{37}$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \eta_i = 0 \tag{38}$$

$$\frac{\partial L}{\partial \xi_i^*} = C - \alpha_i^* - \eta_i^* = 0 \tag{39}$$

$$\frac{\partial L}{\partial \alpha_i} = y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b - \epsilon - \xi_i \leqslant 0$$

$$\frac{\partial L}{\partial \alpha_i^*} = -y_i + \langle \mathbf{w}, \mathbf{x}_i \rangle + b - \epsilon - \xi_i^* \leqslant 0$$

$$\frac{\partial L}{\partial \eta_i} = \sum_{i=1}^{l} \xi_i \leqslant 0$$

$$\frac{\partial L}{\partial \eta_i^*} = \sum_{i=1}^{l} \xi_i^* \leqslant 0.$$

Also, based on the KKT conditions, the products of the Lagrange multipliers and the constraints are equal to zero [4, 62]:

$$\alpha_i(-y_i + \langle \mathbf{w}, \mathbf{x}_i \rangle + b + \epsilon + \xi_i) = 0 \tag{40}$$

$$\alpha_i^*(y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b + \epsilon + \xi_i^*) = 0 \tag{41}$$

$$\eta_i \xi_i = 0 \tag{42}$$

$$\eta_i^* \xi_i^* = 0 \tag{43}$$

Substituting Eq. (36), (37), (38) and (39) into Eq. (35) gives the dual optimization problem [62]:

$$\begin{aligned} &\text{maximize} \quad -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \mathbf{x}_i, \mathbf{x}_j \rangle - \epsilon \sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) \\ &\text{subject to} \quad \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \quad \text{and} \quad \alpha_i, \alpha_i^* \in [0, C] \end{aligned}$$

Based on the Eq. (37), the mathematical expression of the weight vector **w** could be written as follows:

$$\mathbf{w} = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\mathbf{x}_i. \tag{44}$$

Also parameter b needs to be computed and for this case several remarks could be made. Firstly, by giving the mathematical expressions of the Lagrange multipliers η_i and η_i^* from the Eq. (38) and (39) and substituting them into the Eq. (42) and (43) gives:

$$(C - \alpha_i)\xi_i = 0 \tag{45}$$

$$(C - \alpha_i^*)\xi_i^* = 0. \tag{46}$$

From the Eq. (45) and (46) it could be noticed that only sample pairs $(\mathbf{x}_i, y_i)$ with α_i or α_i^* equals to C lie outside the ϵ-insensitive tube because then the parameter ξ_i or ξ_i^* is bigger than zero and equality is satisfied. It follows that, when the sample pair is outside of the tube then α_i or α_i^* is nonzero but they cannot both be zero because then it indicates that the sample pair belongs to the lower and upper boundary, which is not possible [4]. Therefore, based on Eq. (40), (41), (45) and (46) it could be concluded that:

$$- y_i + \langle \mathbf{w}, \mathbf{x}_i \rangle + b + \epsilon \geqslant 0 \quad \text{and} \quad \xi_i = 0 \quad \text{if } \alpha_i < C$$

$$- y_i + \langle \mathbf{w}, \mathbf{x}_i \rangle + b + \epsilon \leqslant 0 \quad \text{and} \quad \xi_i = 0 \quad \text{if } \alpha_i > 0$$

Analogous conclusion could be made with α_i^* and summarizing both analysis on α_i and α_i^* we have:

$$max\{y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - \epsilon | \alpha_i < C \text{ or } \alpha_i^* > 0\} \leqslant b \leqslant min\{y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - \epsilon | \alpha_i > 0 \text{ or } \alpha_i^* < C\}. \quad (47)$$

By taking some $\alpha_i, \alpha_i^* \in (0, C)$ the inequalities in the Eq. (47) become equalities and it allows to solve the issue of computing b. [62]

A final note of the linear SVR could be made by substituting the mathematical expression of **w** given in the Eq. (44) into the linear regression function defined in the Eq. (32) that gives:

$$f(\mathbf{x}) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \langle \mathbf{x}_i, \mathbf{x} \rangle + b. \quad (48)$$

Conjointly, Eq. (44) and (48) refer to the so-called Support Vector expansion. It could be noticed that **w** can be completely described as a linear combination of the data patterns $\mathbf{x}_i$ as well as the complete algorithm can be described in terms of dot products between the data. Also, based on the Eq. (40) and (41), as it was already noted, the Lagrange multipliers may be nonzero only for the data samples outside the ϵ-tube ($|f(\mathbf{x}_i) - y_i| \geqslant \epsilon$) while for the samples inside the tube ($|f(\mathbf{x}_i) - y_i| < \epsilon$) the dual variable α_i, α_i^* has to be zero such that the KKT condition would be satisfied because the second factors in these equations of the condition are nonzero. [62] Therefore, the sums $(\alpha_i - \alpha_i^*)$ in the Eq. (48) are nonzero only for the data samples outside the ϵ-insensitive tube and they are involved into the final decision function by eliminating those which are equal to zero and do not make sense. The data patterns that have nonzero Lagrange multipliers are called the support vectors which "support" the definition of the approximate function and allows us to rewrite the the linear regression function defined in the Eq. (48):

$$f(\mathbf{x}) = \sum_{i=1}^{l} (\alpha_k - \alpha_k^*) \langle \mathbf{x}_i, \mathbf{x}_k \rangle + b \quad k = 1, 2, \ldots, n, \quad (49)$$

where $\mathbf{x}_k$ denotes the support vector and n is the number of support vectors [71].

The formulation of the optimization problem in the dual form is very advantageous because then the input vectors are simply multiplied as dot products and the complexity of a function's representation by support vectors is independent of the dimensionality of the input space $\mathcal{X}$ [62, 71]. These observations are useful in dealing with the nonlinear SVR.

By solving the nonlinear regression problem using SVR, the inputs **x** are nonlinearly mapped into a high dimensional feature space $\mathcal{F}$ by a nonlinear function $\phi(\mathbf{x})$ ($\phi : \mathcal{X} \rightarrow \mathcal{F}$), therefore the decision function becomes [62, 71]:

$$f(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b \quad (50)$$

Basically, the main idea and all calculations when analyzing the nonlinear regression problems using SVR are similar to those which were already presented for the linear case, just by replacing all instances of $\mathbf{x}_i$ in calculations with $\phi(\mathbf{x}_i)$. Therefore, only the main concept and differences could be introduced. For example, the convex optimization problem, defined in the Eq. (34) for the nonlinear case could be written as [71]:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ &\text{subject to} \quad \begin{cases} y_i - \langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b \leqslant \epsilon + \xi_i \\ \langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b - y_i \leqslant \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geqslant 0, \quad i = 1, \ldots, l. \end{cases} \end{aligned} \tag{51}$$

It could be noted, that dealing with the nonlinear regression problems by using SVR the optimization problem corresponds to finding as flat as possible function in the feature space, not in input space [62]. Based on the Eq. (51) the dual form of the nonlinear SVR optimization problem can be expressed as [71]:

$$\begin{aligned} &\text{maximize} \quad -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle - \epsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) \\ &\text{subject to} \quad \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \quad \text{and} \quad \alpha_i, \alpha_i^* \in [0, C]. \end{aligned}$$

However, the computation of $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$ in the feature space could be too complex to perform and here the SVR proposes an advantageous solution: to perform the computation in the input space by using a kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$ to yield the inner products in the feature space [71]. As it was mentioned in the literature review, any functions that satisfy Mercer's condition (see more in [62]) can be used as a kernel function and several of the most commonly used kernels in SVR are presented in the Table 4 [14, 71].

Kernel function	**Equation**
Linear	$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j\rangle$
Polynomial	$K(\mathbf{x}_i, \mathbf{x}_j) = [\gamma\langle \mathbf{x}_i, \mathbf{x}_j\rangle + c]^d$
Sigmoid	$K(\mathbf{x}_i, \mathbf{x}_j) = tanh[\gamma\langle \mathbf{x}_i, \mathbf{x}_j\rangle + c]$
Radial basis function (RBF)	$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$
Gaussian	$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\|\mathbf{x}_i, \mathbf{x}_j\|^2/2\sigma^2)$

Table 4. Most commonly used kernel functions.

Finally, by replacing $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$ with the kernel function, the decision function for the nonlinear case defined in the Eq. (50) could be rewritten based on the Eq. (49) [71]:

$$f(\mathbf{x}) = \sum_{i=1}^{l}(\alpha_k - \alpha_k^*)K(\mathbf{x}_i, \mathbf{x}_k) + b.$$

2.6 Forecasting performance metrics

The main idea of the time series forecasting is to make predictions for the future observations by using the certain model. For the purpose of assessing the quality of forecasting models the most important step is to evaluate the accuracy of predictions [59]. Typically, different forecasting performance metrics are used to evaluate how close the predicted values are to the expected/real values of time series. The literature review shows that the most commonly used are these absolute and squared forecasting error measures: Mean Absolute Error (MAE), Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) which are calculated using the following formulas respectively [31, 59]:

$$MAE = \frac{1}{n}\sum_{t=1}^{n} |x_t - \widehat{x}_t|,$$

$$MSE = \frac{1}{n}\sum_{t=1}^{n} (x_t - \widehat{x}_t)^2,$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n} (x_t - \widehat{x}_t)^2} = \sqrt{MSE}.$$

Here x_t is the measured value of the time series at the time t, $\widehat{x}_t$ is the predicted value of x_t and n denotes the number of observations/predictions.

Also very useful are scaled error measures and one of them is the Mean Absolute Scaled Error (MASE). Differently from the MAE, MSE and RMSE, which are scale-dependent metrics, the result of MASE is independent of the scale of the data [31]. By using this metric the errors are scaled based on the in-sample MAE from the naïve (random walk) forecasting method, in which the one-step prediction for the each observation, used for the model fitting (in-sample), is equal to the actual value of the previous observation [31]. MASE is given by using the following formula:

$$MASE = \frac{1}{n}\sum_{t=1}^{n}\left(\frac{|x_t - \widehat{x}_t|}{\frac{1}{m-1}\sum_{i=2}^{m}|x_i - x_{i-1}|}\right) = \frac{MAE}{\frac{1}{m-1}\sum_{i=2}^{m}|x_i - x_{i-1}|}.$$

Here m denotes the number of observations in-sample and n is the number of predictions. The value of MASE is less than one if the predictions obtained using the constructed model are better than the average one-step naïve forecast computed in-sample and, conversely, it is greater than one if the predictions are worse than the average one-step naïve forecast computed in-sample [31].

3 Experimental results

In the experiment, different models are used for the nonstationary and nearly nonstationary time series forecasting. The performance of the ARIMA is benchmarked against the performance of MLP, RNN and SVR models. MLP network was trained with the backpropagation learning algorithm and backpropagation through time was used as the training algorithm for RNN. The objective of this experiment is to verify the ability of different models to make one-step predictions for the nonstationary and nearly nonstationary time series and make a comparison of the performance of models under consideration.

The literature review shows that ML models could be applied for the nonstationary time series forecasting and, in most cases, could make quite accurate predictions. It should be kept in mind that the nearly nonstationary time series also exist but forecasting of such time series is not widely studied in the literature and is the area of interest. A thorough search of the relevant literature yielded no related works investigating specifically the nearly nonstationary time series forecasting using ML models. Therefore, the experiment is performed to apply several ML models for the nonstationary and nearly nonstationary time series forecasting in an attempt to determine whether ML methods could give quite accurate predictions for both nonstationary and nearly nonstationary time series.

Simulated time series are used for the purpose to analyze the influence of nonstationarity for the time series forecasting. The idea is to check whether ML models could deal with nearly nonstationary time series similarly as well as with nonstationary time series. The comparison of the performance of models is accomplished to identify which ML model could obtain more accurate predictions for the simulated time series and to check if these models could show better or at least as good performance in forecasting as a traditional econometric model. At the end of the experiment, the application of the real world data is used for the purpose of making more general conclusions about the forecasting performance of models when they are used to predict the time series which follow a random walk.

3.1 The performance of models of the simulated time series forecasting

First of all, in the experiment, one simulated nonstationary and a nearly nonstationary time series were used to construct the appropriate architecture of models and make the first comparison of the performance of models under consideration. The first order autoregressive (AR(1)) process with the coefficient β equal to 1 and 0.99, representing the nonstationary and nearly nonstationary time series respectively, were generated. Random number generator was set to the fixed value for the purpose to analyze a single AR(1) realization. The length of both simulated time series is equal to 3000 data points in time. For the purpose of demonstration, Figure 5 shows both simulated AR(1) processes, used at the beginning of the experiment.

Each simulated time series was divided into three data sets. 80% of all data was used for the training and validation of the network by taking 70% of this data subset for the training, development and fitting the model parameters and using the remaining part of the data subset as validation data to provide an evaluation of a model fit on the training data set while tuning model hyperparameters. The test sample consisted of 20% of all data and was used to evaluate the final model with optimal parameters.

After splitting the data all features in the data subsets were transformed by using the so-called Min-Max scaling to translate each feature to a given range between zero and one and thus to make faster and more stable learning process. This scaling is typically done using the following equation:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}},$$

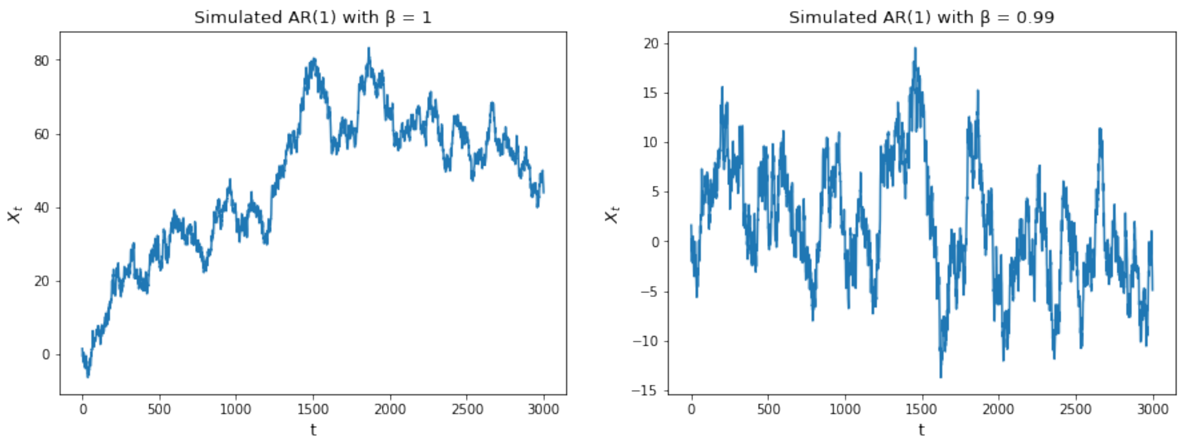


Figure 5. Simulated nontastionary and nearly nonstationary AR(1) time series.

where X is the real value of a feature, X_{scaled} is the normalized value of X, X_{min} and X_{max} are minimum and maximum values respectively in the range of features. It should also be mentioned that in the experiment 2 previous values of the time series are used to predict a future value.

After the preparation of data for further use, each of the ML models intended for the time series forecasting was constructed with several different combinations of parameters by trying to find more appropriate architecture of the model and get more accurate predictions for the AR(1) processes. Three metrics (MSE, RMSE and MAE) were used for the forecasting performance evaluation. Additionally, for the final evaluation after the selection of the optimal parameters, metric MASE was included, to check, whether the constructed model performs better in forecasting compared to the naïve method, which simply set all forecasts for the in-sample data (or training data) to be the value of the last observation.

The experiment is divided into two parts by analyzing the performance of forecasting models for simulated nonstationary and nearly nonstationary time series separately. In each part after the comparison of models, used for the forecasting of a single AR(1) realization, the performance of models was evaluated using more AR(1) realizations. For the nonstationary and nearly nonstationary time series forecasting more realizations of AR(1) were generated by changing the fixed value of a random number generator. Additionally, for the purpose of a deeper analysis of nearly nonstationary time series forecasting, the effect of changing the value of parameter β was examined. Due to the fact that the construction of the optimal model architecture is really time consuming and the experiment is not based on a case study, the same parameters, previously determined as optimal for the single AR(1) realization, were used for all of the AR(1) realizations forecasting. Using the same parameters could also show the influence of the architecture of the model for the time series forecasting which is another interesting part of the experiment. Although, the validation data set was used to evaluate a model fit on the training data set while tuning model hyperparameters and all optimal parameters were already defined, it was decided to keep the same structure of data splitting and divide each AR(1) realization into three data sets (training, validation and testing).

Finally, by taking maximum, minimum and average values of metrics, used to evaluate the model performance of each time series realization forecasting, the experimental results are summarized and the comparison of different models is presented.

3.1.1 Nonstationary time series forecasting

The AR(1) time series with the coefficient $\beta = 1$, illustrated on the left of Figure 5, was used for the first comparison of the performance of forecasting models. Firstly, for the purpose to find optimal parameters, the time series was divided into three parts. The training data set consisted of the first 1680 data points, for the validation further 720 data points were used and the testing data set contained the rest 600 observations. ML models were trained using several combinations of parameters and the one with which the smallest errors on the validation data were obtained was considered as optimal. All ML models require to define the number of inputs which is equal to 2 because it was already mentioned that 2 previous values of the time series are used to find the prediction of a further value. Consequently, the number of outputs is equal to 1. The selection of other parameters depending on each model and the performance of forecasting the analyzed AR(1) time series are described below.

3.1.1.1 Hyperparameters tuning of Multilayer Perceptron (MLP)

The literature review shows that the three-layer MLP with a sufficient number of hidden nodes could give appropriate results in nonstationary time series forecasting, consequently, it was decided to use only one hidden layer in the network. The construction of the MLP network architecture requires defining a number of hidden nodes and there was chosen to try the performance of the network with 2, 4, 5, 6, 8 and 10 hidden nodes. To implement backpropagation it is also important to define a learning rate and through the experiments was noticed that using a bigger value than 0.0001 such as 0.01, 0.001 or even 0.0005 the squared error defined in Eq. (12) starts to increase very fast at each iteration that could imply overshooting local minimum. Therefore, it was decided to use a learning rate equal to 0.0001. Initial weights and biases were set to random numbers with a uniform distribution over the interval [0,1). The logistic sigmoid and ReLU functions were used as the nonlinear activation functions in the hidden layer and the linear/identity function was used in the output layer, as presented in Section 2.3.2. By trying to find out how many forward and backward passes we need for the training of the network was noticed that the training and validation loss decrease when the number of iterations increases, thus more iterations could help to get more accurate predictions. However, it is known that a large number of iterations could cause overfitting when the model starts to memorize the training data by showing a good fit on them while it does not generalize well on new, unseen data. Usually, as the number of iterations increases, at first, both the training loss and the validation loss start decreasing, but after a certain point, the validation error starts to increase while the training error continues to decrease, and then it is assumed that the overfitting appears. Therefore, was experimented to use 10000 iterations and then, after finding the most appropriate parameters of the network and considering the behavior of train and validation loss functions, to decide whether a such number of iterations is sufficient or whether it needs to be reduced/increased.

Hence, 12 different architectures of the MLP network were evaluated after the final iteration was reached. The evaluation of the final predictions obtained using MLP network with different activation functions and a number of hidden nodes was made using 3 metrics (MSE, RMSE and MAE) on the scaled training and validation data. Figure 6 shows all combinations of parameters used for the construction of the MLP network and values of metrics used to measure the forecasting performance in the validation data set. It could be noticed that using ReLU activation function and 10 hidden nodes the error obtained on the validation data is really huge compared to other combinations of parameters.

The Table 5 includes combinations of parameters giving 10 smallest values of all performance metrics. It has to be highlighted that in this experiment all included tables representing combinations of models parameters giving smallest values of performance metrics are shown in a sorted order by RMSE, MSE and MAE (computed on the validation data) accordingly. It could be seen from the Table 5, that using ReLU activation function the better performance could be reached than using sigmoid function. It is interesting that later, when more realizations of time series were predicted using ReLU in the hidden layer, it was noticed, that the errors obtained on several time series were really large compared to those obtained using sigmoid activation function. However, due to the fact that in this experiment the optimal parameters are selected based on the single time series

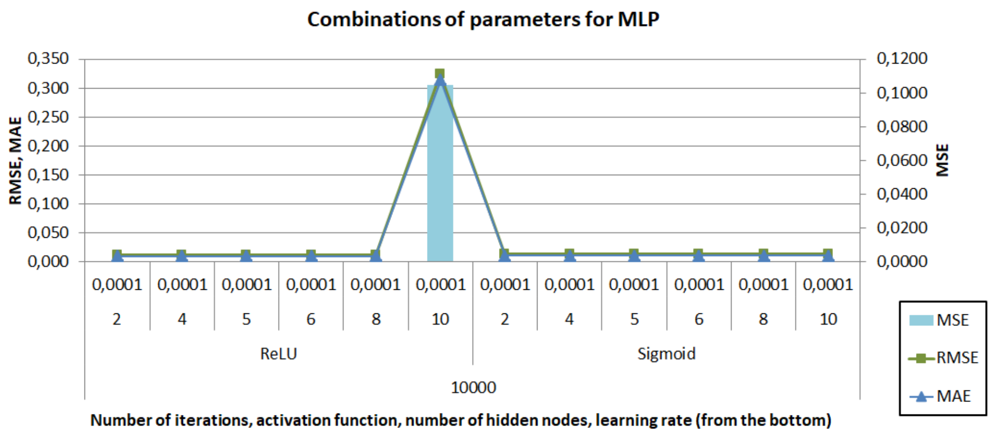


Figure 6. All combinations of parameters used to construct the MLP network for the nonstationary time series forecasting and values of performance metrics calculated on scaled validation data.

realization, it was decided to choose ReLU as the optimal activation function in the MLP network. It could also be noticed from the Table 5 that the network with ReLU function in the hidden layer shows the best performance with 6 hidden nodes. Therefore, the architecture of MLP with ReLU activation function, 2 input nodes, 6 hidden nodes and a single output node is considered as optimal by taking 10000 iterations and a learning rate equal to 0.0001 to train the network.

Activation function	Number of hidden nodes	Learning rate	Number of iterations	Validation data			Train data		
				MSE	RMSE	MAE	MSE	RMSE	MAE
ReLU	6	0.0001	10000	0.000138	0.011760	0.009513	0.000137	0.011690	0.009179
ReLU	4	0.0001	10000	0.000144	0.011996	0.009716	0.000142	0.011907	0.009339
ReLU	8	0.0001	10000	0.000146	0.012074	0.009808	0.000142	0.011905	0.009348
ReLU	2	0.0001	10000	0.000147	0.012111	0.009814	0.000144	0.012020	0.009423
ReLU	5	0.0001	10000	0.000152	0.012328	0.010001	0.000150	0.012251	0.009589
Sigmoid	6	0.0001	10000	0.000166	0.012893	0.010360	0.000158	0.012568	0.009950
Sigmoid	8	0.0001	10000	0.000177	0.013297	0.010714	0.000164	0.012788	0.010153
Sigmoid	4	0.0001	10000	0.000178	0.013353	0.010758	0.000165	0.012851	0.010196
Sigmoid	10	0.0001	10000	0.000179	0.013364	0.010755	0.000166	0.012875	0.010211
Sigmoid	2	0.0001	10000	0.000185	0.013617	0.010964	0.000172	0.013124	0.010404

Table 5. 10 combinations of parameters used to construct the MLP network for the nonstationary time series forecasting giving smallest values of performance metrics on the scaled validation data.

The MSE function was used to ensure that 10000 iterations did not cause overfitting. For the purpose to more clearly display the behaviour of MSE function when the decrease of training and validation error is quite fast and then slows down, line chart is splitted and on the left of Figure 7 the training and validation loss after 3-500 iterations are shown and on the right the remaining part of loss functions is presented. The training error after the first two iterations is very large (1.631 and 0.555) but after that backpropagation helps to adjust weights and biases of the network and then after the third iteration the loss suddenly decreases until it reaches 0.011. Validation loss also has a huge decrease after the second iteration (from 1.052 to 0.034) then after the third iteration it decreases until 0.011. After the fourth iteration validation loss slightly increase until 0.012 and later starts to steadily decrease. The fluctuation of loss function at the beginning of the training is acceptable because the network requires more time to learn and is not able to generalize well. Due to the large changes of loss values first two iterations are not included into the Figure 7 for the more clear demonstration. Generally, we could see that both training and validation losses steadily decrease and at the end of iterating the gap between these functions becomes infinitesimal (about 0.000001). Therefore, it was decided that 10000 iterations does not cause overfitting and allows to reach quite small errors of predictions.

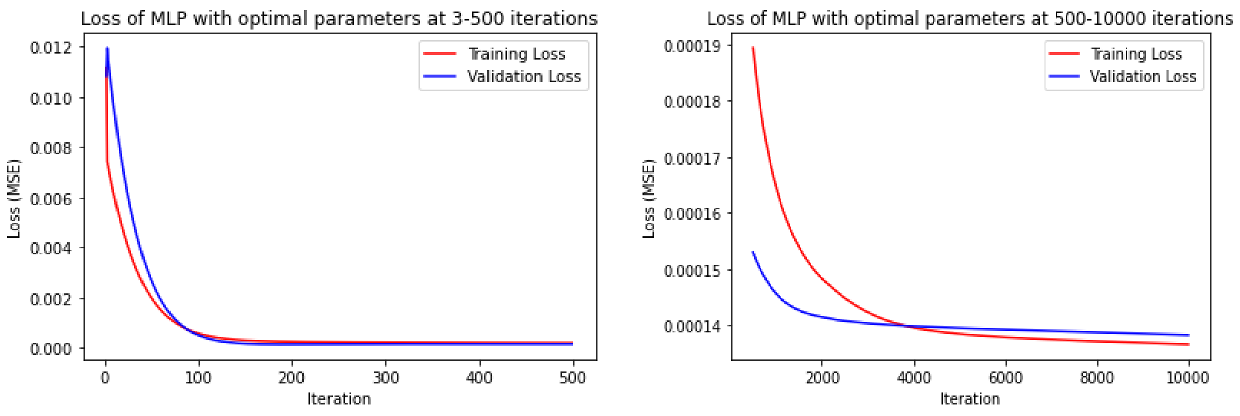


Figure 7. The training and validation loss (MSE) of the MLP with the optimal parameters after each of 3-500 iterations (on the left) and after the 500-10000 iterations (on the right). All calculation are made on the scaled data.

After the constructing an optimal architecture of MLP with BP and selecting other important parameters, test data was used to evaluate the final model. The Figure 8 shows the unscaled predicted data and actual values of AR(1) with the coefficient $\beta = 1$ by showing training, validation and test data sets separately. The Figure 9 is included to take a closer look at predicted values. It could be seen that the predicted values are quite accurate, although a small shift could be seen and the model does not ideally handles all spikes in the data.

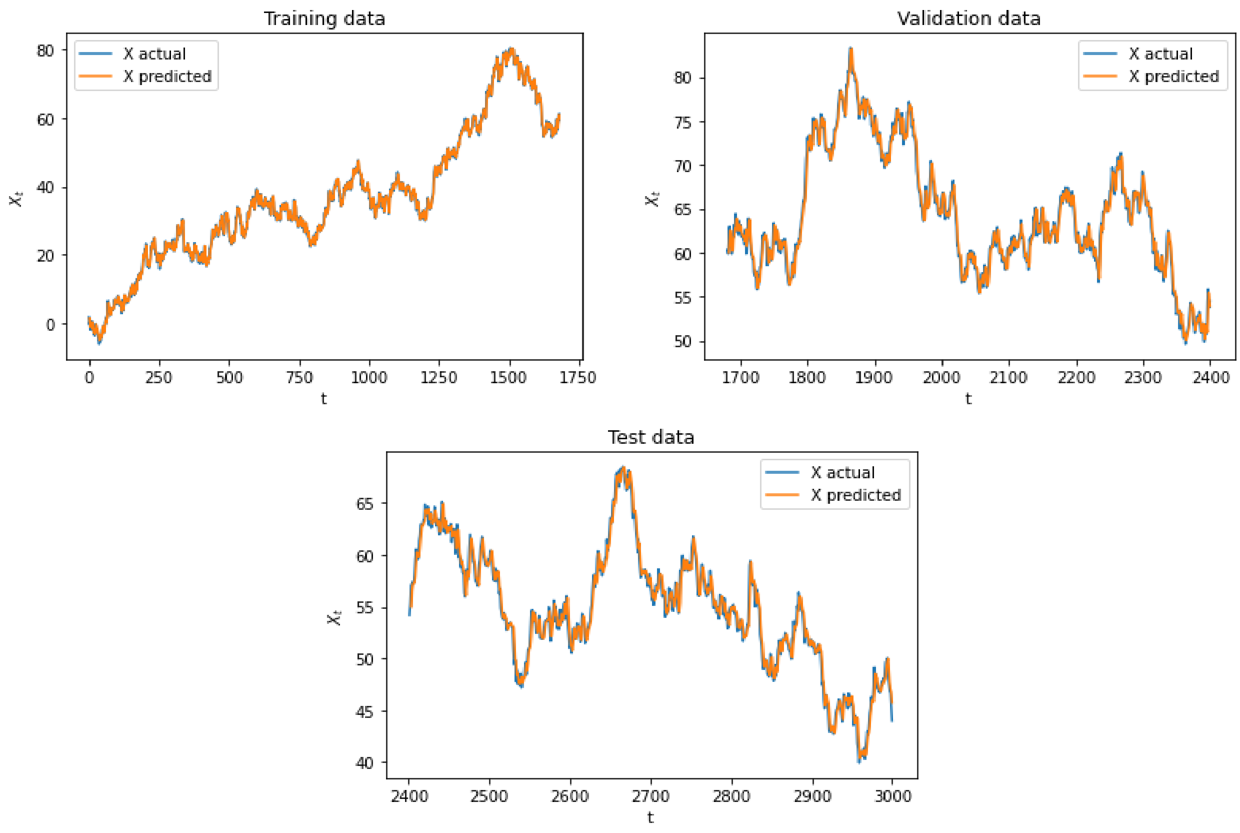


Figure 8. Actual and predicted nonstationary AR(1) time series with $\beta = 1$ using MLP with optimal parameters (ReLU activation function, 6 hidden nodes, learning rate equals to 0.0001 and 10000 iterations).

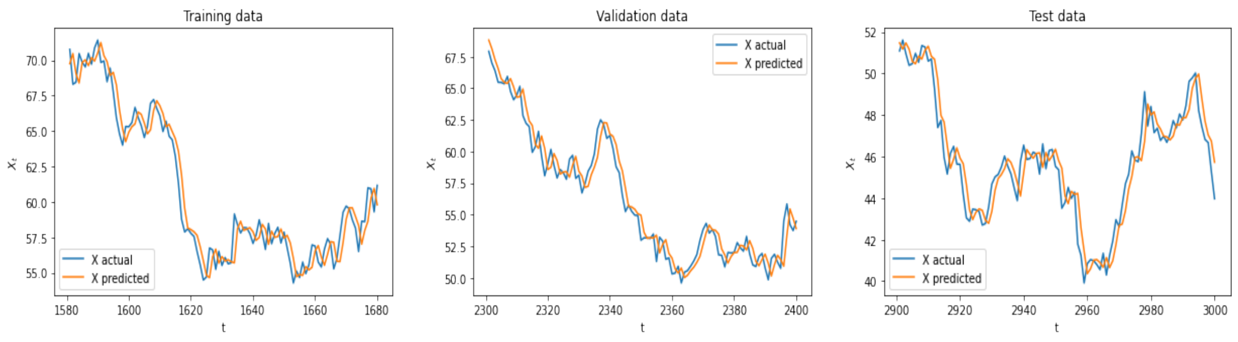


Figure 9. Last 100 actual and predicted values of training, validation and test data sets using MLP for the forecasting.

3.1.1.2 Hyperparameters tuning of Recurrent Neural Network (RNN)

In this experiment for the time series forecasting it was decided to use the three layer RNN and to check whether the ability of the RNN to keep in memory previous computations helps to achieve better forecasting performance and outperform the three layer MLP or whether both networks provide similar results. The architecture of RNN is very similar to MLP and, as in MLP, there are 4 most important parameters that have to be defined: activation function in the hidden layer (in the output layer, as in MLP, the linear/identity function is used), number of hidden nodes, size of learning rate and number of iterations required to train the network. There was decided to use the same activation functions (ReLU and sigmoid) and numbers of hidden nodes (2, 4, 5, 6, 8, 10) as in MLP network. Through the experiments it was noticed, that for the RNN, a slightly bigger than 0.0001 learning rate is more preferable and helps more faster achieve the better performance of forecasting, therefore, was chosen to use two values of learning rate equal to 0.01 and 0.001. For the purpose of checking, whether the predictions obtained are more accurate with bigger or smaller number of iterations, it was decided to use 4500 and 6000 iterations and then after looking at the validation and training loss functions to conclude if the number of iterations should be increased or decreased.

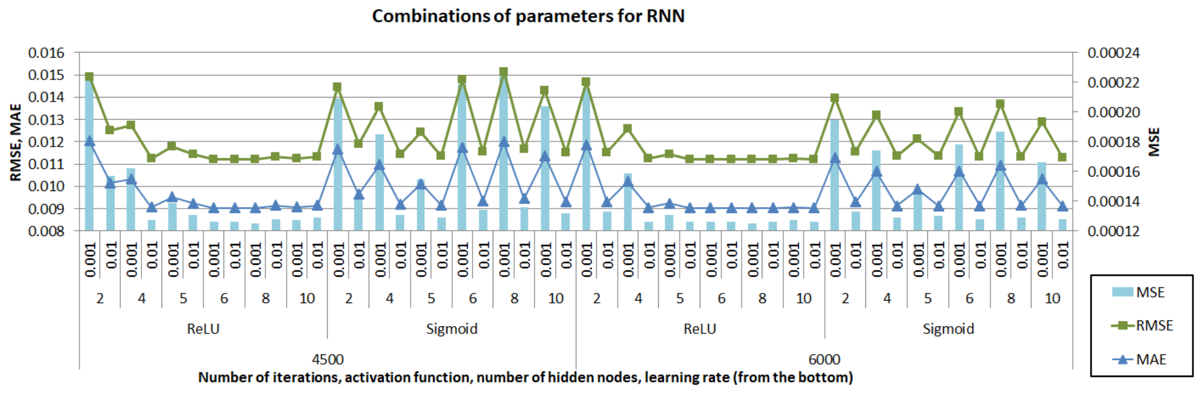


Figure 10. All combinations of parameters used to construct the RNN for the nonstationary time series forecasting and values of performance metrics calculated on scaled validation data.

The Figure 10 shows all combinations of parameters used for the construction of RNN and the values of three metrics used to measure the forecasting performance in validation data set. It could be noticed that the network with only 2 hidden nodes and a smaller learning rate (0.001) is inappropriate combination regardless of which activation function and number of iterations are used. Also, using sigmoid activation function in the hidden layer, the bigger value of learning rate helps to achieve better performance of forecasting, while by using ReLU function and more than 5 hidden nodes, the difference between errors obtained using different values of learning rate is not so obvious. It could be seen that the increase of the number of iterations has a bigger influence for the sigmoid activation function compared to ReLU function, but not in all combinations of parameters with sigmoid function, more iterations help to achieve better forecasting performance. In general, smaller errors between actual (scaled) and predicted validation data could be get using ReLU activation function and 6-10 hidden nodes or 5 hidden nodes and a bigger learning rate or using sigmoid activation function with a bigger learning rate.

The Table 6 is included to show 10 combinations of parameters giving smallest errors on the scaled validation data. It could be seen that values of metrics do not change very significantly between these 10 most optimal combinations. As it was already noticed from the Figure 10, usually, the increase of the number of iterations does not significantly affect the errors obtained using the ReLU activation function and same architecture of the network and the change of errors after rounding is not even seen or is very small. As in MLP network, the better choice is to use ReLU activation function in the hidden layer by comparing it to the sigmoid function. Values of MSE and RMSE are smallest when the ReLU activation function, 8 hidden nodes and learning rate equals to 0.001 are used in the network while MAE shows that the better choice is to use 10 hidden nodes and a bigger learning rate (0.01) taking more iterations for the training of the network. However, it was decided to rely on the values of MSE and RMSE and choose less hidden nodes (8) in the network and less iterations (4500).

Activation function	Number of hidden nodes	Learning rate	Number of iterations	Validation data			Train data		
				MSE	RMSE	MAE	MSE	RMSE	MAE
ReLU	8	0.001	4500	0.000125	0.011200	0.009027	0.000127	0.011269	0.008963
ReLU	8	0.001	6000	0.000125	0.011200	0.009028	0.000126	0.011236	0.008933
ReLU	10	0.01	6000	0.000126	0.011207	0.009024	0.000126	0.011213	0.008895
ReLU	6	0.001	4500	0.000126	0.011208	0.009031	0.000126	0.011209	0.008889
ReLU	6	0.001	6000	0.000126	0.011208	0.009031	0.000126	0.011206	0.008883
ReLU	6	0.01	6000	0.000126	0.011210	0.009034	0.000126	0.011213	0.008892
ReLU	6	0.01	4500	0.000126	0.011214	0.009035	0.000126	0.011213	0.008889
ReLU	5	0.01	6000	0.000126	0.011214	0.009034	0.000126	0.011214	0.008894
ReLU	8	0.01	6000	0.000126	0.011218	0.009031	0.000126	0.011209	0.008887
ReLU	4	0.01	6000	0.000126	0.011235	0.009046	0.000126	0.011227	0.008911

Table 6. 10 combinations of parameters used to construct the RNN network for the nonstationary time series forecasting giving smallest values of the performance metrics on the scaled validation data.

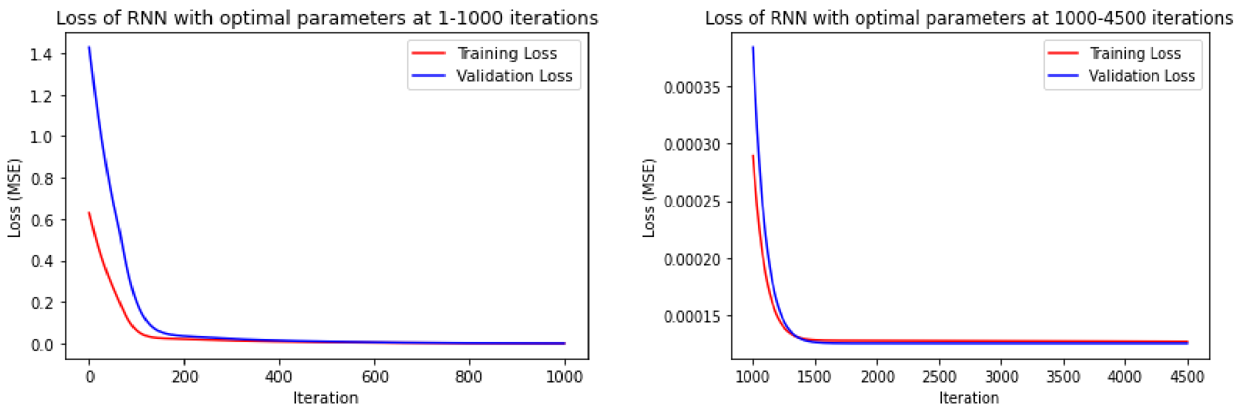


Figure 11. The training and validation loss (MSE) of the RNN with the optimal parameters after each of 1-1000 iterations (on the left) and after the 1000-4500 iterations (on the right). All calculation are made on the scaled data.

When there was noticed that the increase of number of iterations from 4500 to 6000 does not change values of errors very significantly, it was decided to look how do the MSE values, calculated on the validation and training data, change after each of 4500 iterations. From the Figure 11 it could be seen that both training and validation loss functions are decreasing and that could be treated as there is no overfitting because the validation loss does not start to increase at the same point. However, after looking closer at the values of validation loss it was noticed that using less iterations and, thus, making the training process shorter and less computationally expensive, similarly small value of error (MSE) could be obtained as using all 4500 iterations for the training of the network. It was calculated that through the 1000-2000 iterations the decrease of value of MSE was equal to about 0.000258 while through the 2000-4500 iterations the decrease was incredibly small (about 0.000000015). Therefore, it was decided that 2000 iterations could be enough to get quite small errors between the predicted and real values of validation data. For the purpose of confirming that, the values of three metrics, obtained using the selected optimal parameters and 4500 iterations for the training, were compared to those which were obtained taking 2000 iterations. From the Table 7 we could see that, by taking much less iterations, the values of error functions, calculated after the final iteration was reached, are very similar. The value of RMSE slightly increased but the value of MAE is even smaller than using more iterations, thus indicating that the decision of using the smaller number of iterations for the training was appropriate and useful.

Activation function	**Number of hidden nodes**	**Learning rate**	**Number of iterations**	**Validation data**			**Train data**		
				MSE	**RMSE**	**MAE**	**MSE**	**RMSE**	**MAE**
ReLU	8	0.001	4500	0.000125	0.011200	0.009027	0.000127	0.011269	0.008963
ReLU	8	0.001	2000	0.000125	0.011201	0.009025	0.000128	0.011304	0.008996

Table 7. The comparison of values of the performance metrics calculated on the scaled validation data using the optimal parameters of RNN and changing the number of iterations, required for the training of the network.

The Figures 12 and 13 show real and predicted values of nonstationary AR(1) time series using RNN. It could be noticed that predictions are quite accurate but, similarly as using MLP, the small shift could be noticed and it looks like the network predicts future value very close to the previous value.

3.1.1.3 Hyperparameters tuning of Support Vector Regression (SVR)

SVR has three important factors that affect its quality and should be defined before applying model for the time series forecasting: the penalty parameter C, the non-sensitive loss ϵ and the kernel function with its parameters. In this experiment it was decided to use sigmoid, linear and radial basis (RBF) kernel functions. The values of C and ϵ were defined as 2^i, where $i = 1, \ldots, 9$, and 0.1^j, where $j = 2, \ldots, 5$, respectively. Also by using sigmoid and RBF kernels there was required to define kernel parameter γ and there was chosen to use 0.1^j, where $j = 1, \ldots, 4$, as its value. Figure 14 is included here to show, how changing the selected values of kernel function, C and ϵ affect the values of performance metrics, calculated on validation data set.

It could be noticed from the Figure 14 that by using linear kernel with a bigger value of ϵ, the least values of metrics are obtained compared to those which are given using a smaller value of ϵ and the change of C does not very significantly affect the forecasting performance. When the RBF or sigmoid kernel is used, from the Figure 14 it looks like the value of C becomes very important. We could see that the bigger value of C is a better choice by combining it with RBF kernel, while by using sigmoid kernel, oppositely, the smaller value of C gives the better forecasting performance. However, it should be emphasized that the change of value of sigmoid and RBF kernels parameter γ here makes a biggest influence for the obtained results. Through the experiments it was noticed that, using RBF kernel with a smaller value of C and a value of γ bigger than 0.0001 the error

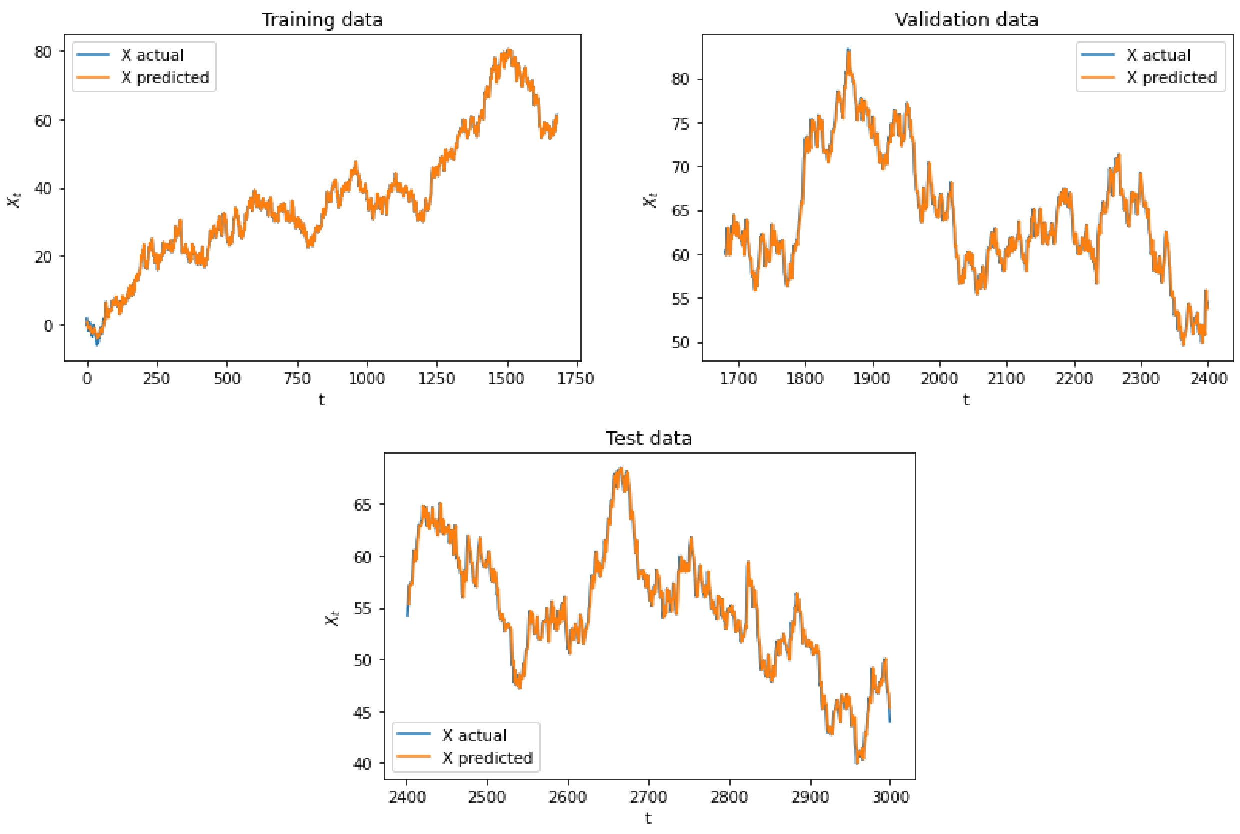


Figure 12. Actual and predicted nonstationary AR(1) time series with $\beta = 1$ using RNN with optimal parameters (ReLU activation function, 8 hidden nodes, learning rate equals to 0.001 and 2000 iterations).

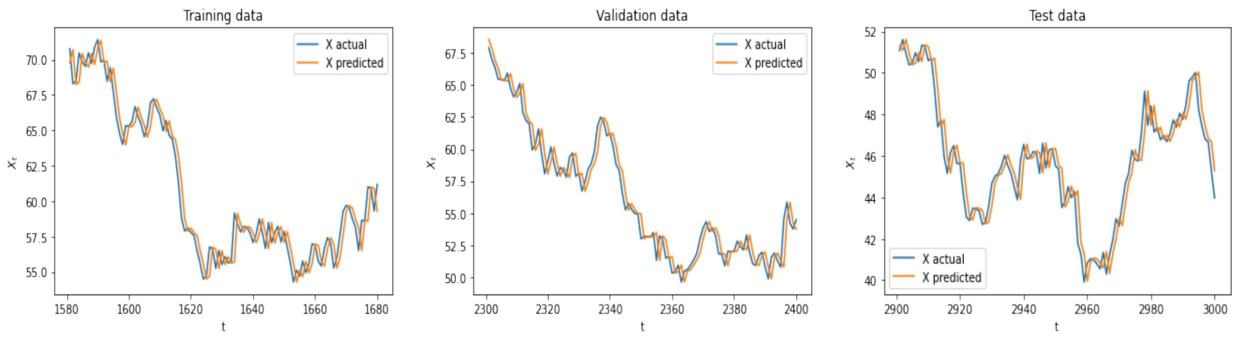


Figure 13. Last 100 actual and predicted values of training, validation and test data sets using RNN for the forecasting.

obtained are quite small and comparable to those which are given by using a bigger value of C. Similarly, by using sigmoid kernel with a bigger value of C and a value of γ smaller than 0.1 also, in most cases, quite accurate predictions could be made.

For the purpose of concentrating more on the most optimal combinations of SVR parameters, the Table 8 is included to show combinations giving the smallest values of performance metrics. It was noticed that different metrics suggest different most optimal combinations of SVR parameters and, due to the fact that all values in this table are sorted in order by RMSE, MSE and MAE accordingly, 15 combinations of parameters were taken by including the one with the smallest value of MAE. Based on the values of RMSE and MSE the most optimal combination consisted of the RBF kernel function with values of C, ϵ and γ equal to 32, 0.01 and 0.1 respectively. It could be seen that the linear kernel with the C equals to or bigger than 4 and ϵ equals to 0.01 also

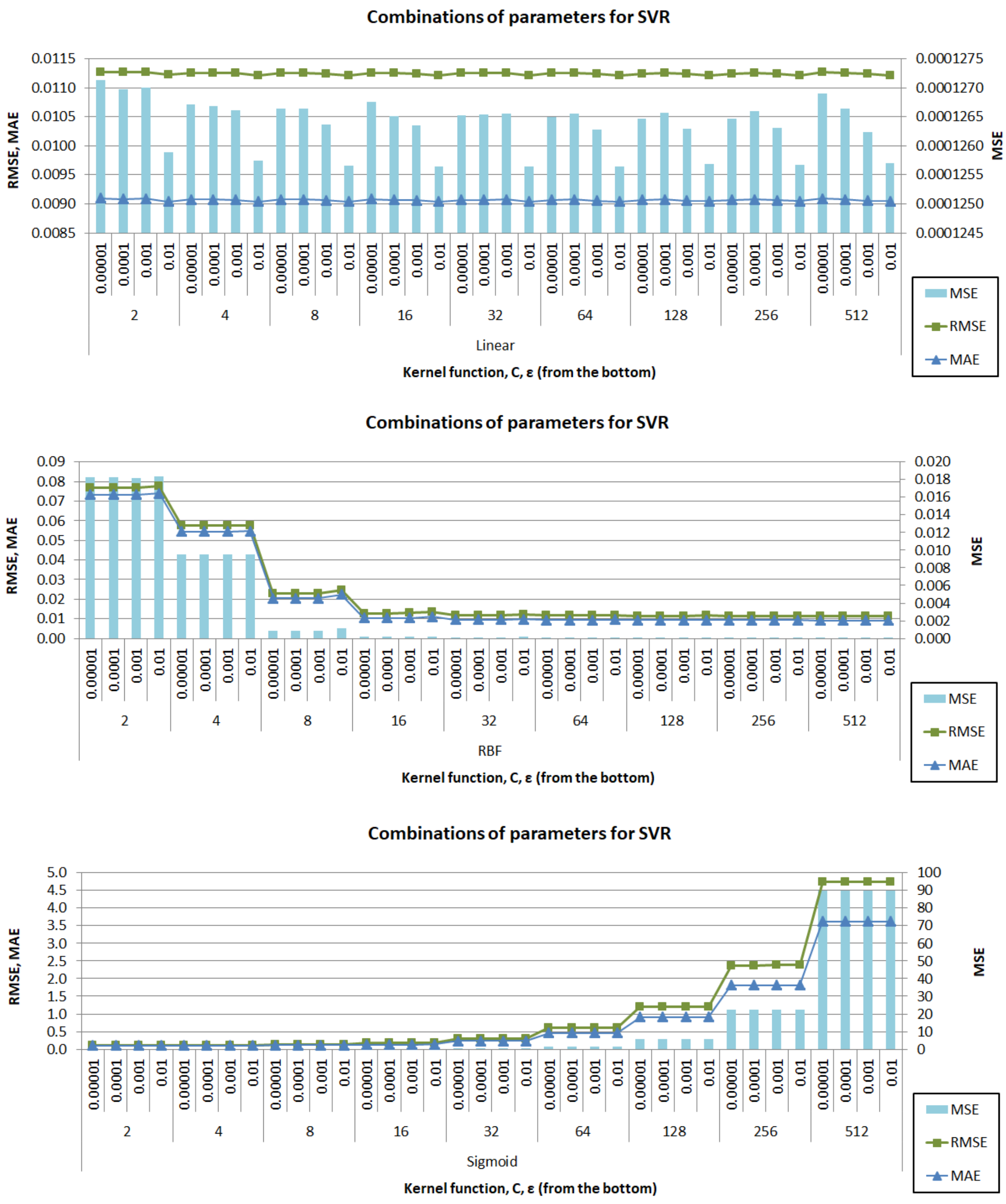


Figure 14. All combinations of three parameters used to construct the SVR for the nonstationary time series forecasting and values of performance metrics calculated on scaled validation data.

show a quite well performance of forecasting. It is interesting that the most optimal combination of parameters, based on the value of MAE, is the sigmoid kernel function with values of C, ϵ and γ equal to 256, 0.01 and 0.01 respectively. Based on the RMSE metric, there are 14 combinations of parameters with a smaller RMSE value than the one with a least value of MAE. For the purpose of making the final decision whether to choose the first combination of parameters in the Table 8 or the fifteenth, it was decided to calculate values of MASE additionally. For the first combination the value of MASE was equal to 1.013706 while for the fifteenth it was a slightly smaller (1.013563) that indicates the superiority of this combination against the first one. Therefore, the combination of sigmoid kernel function with values of C, ϵ and γ equal to 256, 0.01 and 0.01 respectively was chosen as optimal. The Figures 15 and 16 are included to graphically depict the real and predicted values of the nonstationary AR(1) time series using SVR with optimal parameters. Similarly, as using RNN, we could notice a small shift between the real and predicted values even though the predictions look quite accurate.

Kernel function	**C**	ϵ	γ	**Validation data**			**Train data**		
				MSE	**RMSE**	**MAE**	**MSE**	**RMSE**	**MAE**
RBF	32	0.01	0.1	0.0001255	0.0112035	0.0090299	0.0001259	0.0112215	0.0089030
Linear	16	0.01		0.0001256	0.0112090	0.0090401	0.0001259	0.0112215	0.0089034
Linear	64	0.01		0.0001256	0.0112090	0.0090397	0.0001259	0.0112221	0.0089037
Linear	32	0.01		0.0001256	0.0112093	0.0090406	0.0001259	0.0112226	0.0089045
Linear	8	0.01		0.0001257	0.0112098	0.0090403	0.0001259	0.0112208	0.0089011
Linear	256	0.01		0.0001257	0.0112104	0.0090432	0.0001260	0.0112236	0.0089057
Linear	128	0.01		0.0001257	0.0112107	0.0090433	0.0001260	0.0112228	0.0089045
Linear	512	0.01		0.0001257	0.0112113	0.0090449	0.0001260	0.0112236	0.0089060
RBF	64	0.01	0.1	0.0001257	0.0112116	0.0090475	0.0001258	0.0112175	0.0089075
Linear	4	0.01		0.0001257	0.0112134	0.0090401	0.0001259	0.0112196	0.0088940
RBF	128	0.01	0.01	0.0001257	0.0112134	0.0090319	0.0001259	0.0112207	0.0088883
RBF	512	0.01	0.01	0.0001258	0.0112146	0.0090452	0.0001260	0.0112247	0.0088991
RBF	128	0.01	0.1	0.0001258	0.0112176	0.0090572	0.0001259	0.0112191	0.0089122
RBF	256	0.01	0.1	0.0001259	0.0112184	0.0090578	0.0001259	0.0112200	0.0089172
Sigmoid	256	0.01	0.01	0.0001259	0.0112184	0.0090287	0.0001260	0.0112258	0.0089013

Table 8. 15 combinations of parameters used to construct the SVR for the nonstationary time series forecasting giving smallest values of the performance metrics on the scaled validation data.

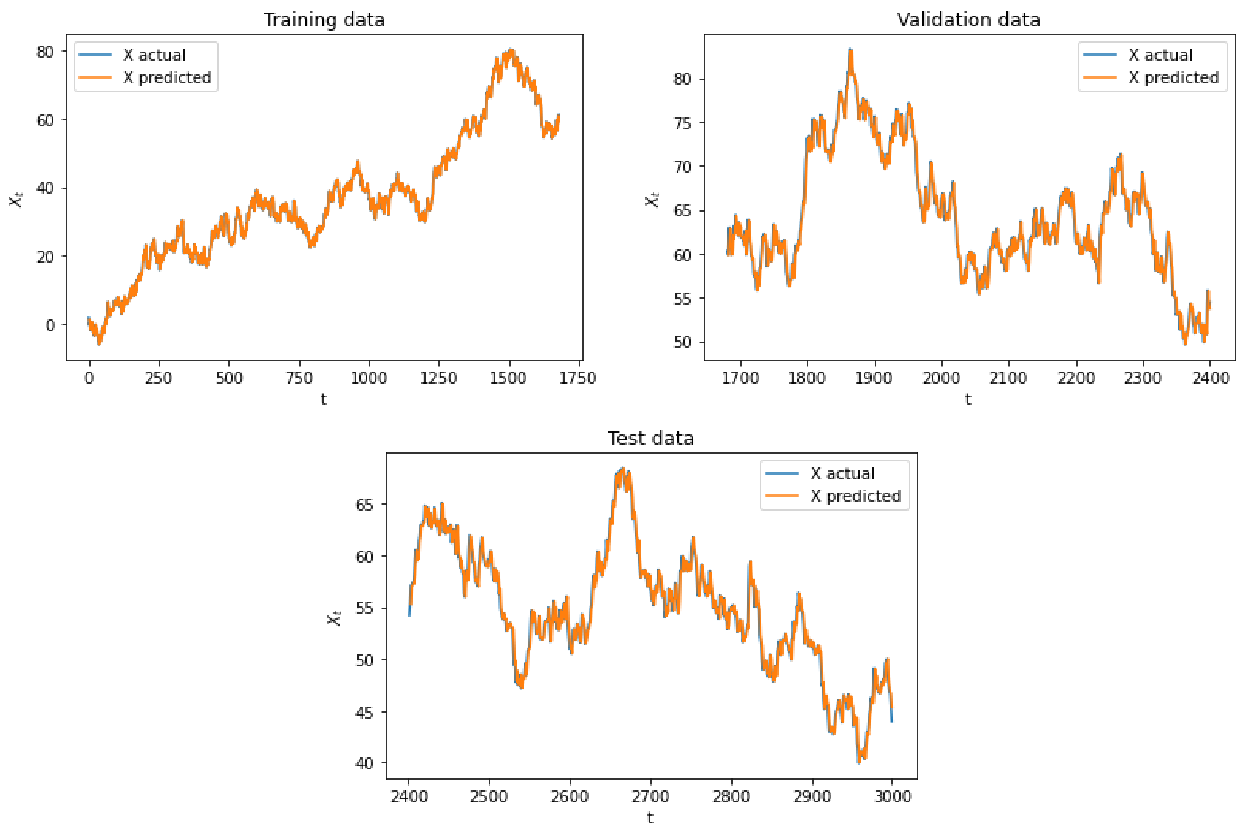


Figure 15. Actual and predicted nonstationary AR(1) time series with $\beta = 1$ using SVR with optimal parameters (sigmoid kernel function with values of C, ϵ and γ equal to 256, 0.01 and 0.01 respectively).

3.1.1.4 Application of ARIMA

ARIMA(p, d, q) model has three parameters that have to be defined: p, d and q. For the nonstationary AR(1) time series forecasting the simplest one and usually used is ARIMA(0,1,0), better known as a random walk model. Therefore, in this experiment was decided to use this model

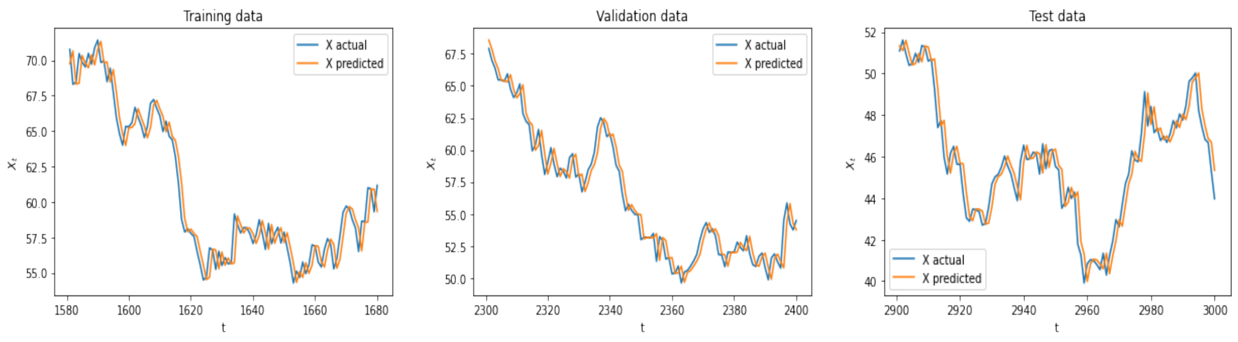


Figure 16. Last 100 actual and predicted values of training, validation and test data sets using SVR for the forecasting.

as a benchmark in an attempt to compare the performance of ML models and the simple statistical forecasting model in the nonstationary time series forecasting.

The model was fitted on the first 2400 data points of the simulated nonstationary time series and after that the predictions were generated for the each element on the test set which consists of the rest 600 data points. Figure 17 is included to show the real test data and predicted values using ARIMA(0,1,0). It could be seen that predictions look quite accurate with a small shift.

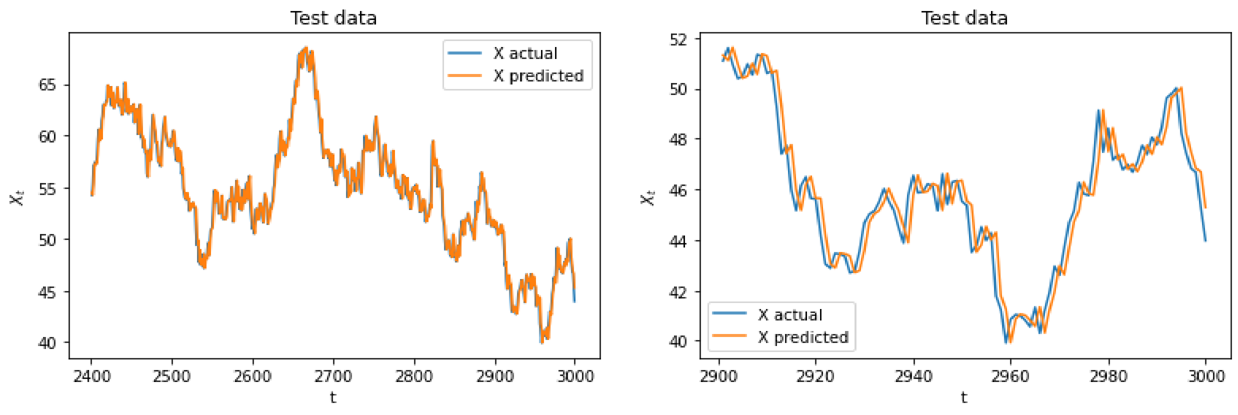


Figure 17. Actual and predicted test data of nonstationary AR(1) time series with $\beta = 1$ using ARIMA(0,1,0) (on the left) by extracting additionally last 100 values (on the right).

3.1.1.5 Comparison of the performance of models of the single AR(1) realization forecasting

After the tuning hyperparameters and applying all selected models for the nonstationary AR(1) time series forecasting, four metrics (MSE, RMSE, MAE and MASE) were used for the final evaluation of the forecasting performance of each model. The main interest of this experiment was to find out how well ML models with the defined optimal parameters could make the one-step predictions on the new unseen data and to compare the forecasting performance of these models with the traditional statistical model used as a benchmark. The real values of the test data set with the obtained predictions were already visualised and from the visual perspective it could be concluded that the predictions obtained using all 4 models under considerations look quite accurate. However, it was noticed that the small shift appears between the real and predicted values.

The related literature shows that, when the process follows a random walk, its previous values contain no useful information for the making predictions of future values [25]. Therefore, the best prediction of x_{t+1} for this process having the minimum MSE is $\hat{x}_{t+1} = x_t$ and this is the most important implication of the random walk hypothesis (see more in [25]). There have been published many works about the forecasting of the stock prices, which usually follow the random walking, and it was confirmed that usually the best forecasting model of such time series is the naïve model which propose to use the previous observation as the prediction of the current data point. One of these works is the article [61] whose authors apply Time Delay Feed Forward Network (TDNNs) and Elman Network (RNN) for the one-step ahead forecasting of the detrended S&P 500 stock market index time series. Authors show that, even though these models were able to make particularly accurate predictions for the modulated sinusoidal signal which frequency ranges from weak low to strong high, unfortunately, they did not succeed to predict the next day value of the stock market index time series better than just taking the value of the previous day, although many combinations of networks parameters have been tried.

In our case the analyzed nonstationary AR(1) time series is the random walk process and even though many different combinations of parameters of ML models under consideration have been experimented by taking more input values or hidden nodes, changing the data splitting proportions or generating a longer time series, the best obtained one-step predictions were always close to the previous values of time series.

In the Table 9 the final results are presented and it could be noticed that SVR shows the best forecasting performance in a comparison with MLP, RNN and ARIMA models. Based on the values of MSE, RMSE and MAE metrics, RNN performs similarly well as SVR and also slightly outperforms ARIMA(0,1,0) model. However, MLP network obtains highest values of errors and the conclusion could be made that not all ML models could outperform traditional statistical model. By looking at the values of MASE we could see that all of them are very close to 1 what indicates that models predict similarly as the naïve model in-sample. Based on the random walk hypothesis these results are not surprising, however, all models generate slightly more accurate predictions for the test data than the naïve model does that for the data in-sample.

	MSE	**RMSE**	**MAE**	**MASE**
MLP	0.965848	0.982775	0.786389	0.986095
RNN	0.907514	0.952635	0.766646	0.961338
SVR	0.905128	0.951382	0.764673	0.958864
ARIMA(0,1,0)	0.910015	0.953947	0.768717	0.959644

Table 9. Final values of metrics used to evaluate the forecasting performance of models on the test data set of the single nonstationary AR(1) realization.

3.1.1.6 Comparison of the performance of models of the 100 AR(1) realizations forecasting

By using the same parameters which were defined as optimal for the single AR(1) realization forecasting, all models were applied for the 100 other realizations of the AR(1) time series. Forecasting performance of models was evaluated for the each of 100 time series using the same four metrics as before (MSE, RMSE, MAE and MASE). The evaluations were made using the predictions of the test data sets. The Table 10 shows minimum, maximum and average values of the performance metrics in the range of 100 evaluations for the each model. It could be seen that the maximum values of metrics by using the MLP and RNN models are incredibly high. It could indicate that the architecture of these networks is very dependent on the specific time series and inappropriate parameters strongly affect the performance of forecasting. However, by looking at the minimum values of metrics, we could notice that the architecture of the neural networks was appropriate not only for the single AR(1) realization, analyzed at the begining of the experiment, and some of 100 time series were predicted also quite accurately. SVR demonstrates its superiority over the neural network models due to the fact that it shows quite well generalization capacity on many different nonstationary time series even by using the same hyperparameters. Therefore, the accuracy of predictions obtained using SVR is not so strongly depended on parameters as applying neural network models. Even though the SVR predicts quite accurately, the ARIMA model outperforms all ML models by looking at the average values of metrics.

Therefore, by concluding the results obtained, it could be noticed that some of ML models could slightly outperform the traditional statistical model in nonstationary time series forecasting just when the specific case is analyzed due to the dependency between the forecasting performance and the model parameters. The selection of ML models parameters is really time consuming but the appropriate combination of parameters helps to obtain quite accurate one-step predictions. However, the predictions of the single AR(1) realization made by using ML models do not very significantly differs from those obtained with the simple ARIMA model which is more simple and easier to use.

	MSE			**RMSE**			**MAE**			**MASE**		
	MIN	**MAX**	**AVG**	**MIN**	**MAX**	**AVG**	**MIN**	**MAX**	**AVG**	**MIN**	**MAX**	**AVG**
MLP	0.9820	734.7689	17.3268	0.9910	27.1066	2.2573	0.7782	24.1288	1.8692	0.9628	30.5597	2.3463
RNN	0.9201	2860.0210	70.3936	0.9592	53.4792	4.1990	0.7600	44.5505	3.5400	0.9384	56.4242	4.4457
SVR	0.9033	1.2786	1.0258	0.9504	1.1307	1.0123	0.7505	0.9074	0.8080	0.9210	1.1492	1.0118
ARIMA(0,1,0)	0.8922	1.1708	1.0088	0.9446	1.0821	1.0040	0.7456	0.8703	0.8009	0.9232	1.0936	1.0026

Table 10. Minimum, maximum and average values of metrics used to evaluate the forecasting performance of models on the test data set of the 100 nonstationary AR(1) realizations.

3.1.2 Nearly nonstationary time series forecasting

The AR(1) time series with the coefficient $\beta = 0.99$, illustrated on the right of Figure 5, was used to clarify whether ML models could give quite accurate predictions for the nearly nonstationary time series and to make a first comparison of forecasting models. Nearly nonstationary time series forecasting was performed analogously as for the nonstationary time series and, first of all, was trying to find the optimal architectures of models. For the construction of each network there was decided to use the same combinations of parameters as were applied for the nonstationary time series forecasting and to check, whether the same combinations are optimal for both nonstationary and nearly nonstationary time series or whether they require different parameters to get more accurate predictions.

3.1.2.1 Hyperparameters tuning of Multilayer Perceptron (MLP)

Through the search of optimal parameters of the MLP network was noticed that there could be found similar tendencies between nonstatinary and nearly nonstationary time series forecasting. Experiment results showed that by using a learning rate bigger than 0.0001 errors obtained of the loss function are very large and the network performs better with a bigger number of iterations as in nonstationary time series forecasting. From the Figure 18 we could see that using the ReLU activation function in the hidden layer and 10 hidden nodes for the nearly nonstationary time series forecasting very inaccurate predictions are obtained as in nonstationary time series forecasting. However, while the combination of ReLU and 8 hidden nodes for the nonstationary time series was an appropriate choice, now we could see that this combination should be eliminated due to the large errors obtained on the validation data set.

The Table 11 shows that using ReLU activation function more accurate predictions could be get than using sigmoid function through the same number of iterations. It is interesting that for the nearly nonstationary and nonstationary time series forecasting using MLP network, the same combination of parameters (ReLU activation function, 6 hidden nodes, learning rate equals to 0.0001 and 10000 iterations) helps to get smallest values of performance metrics.

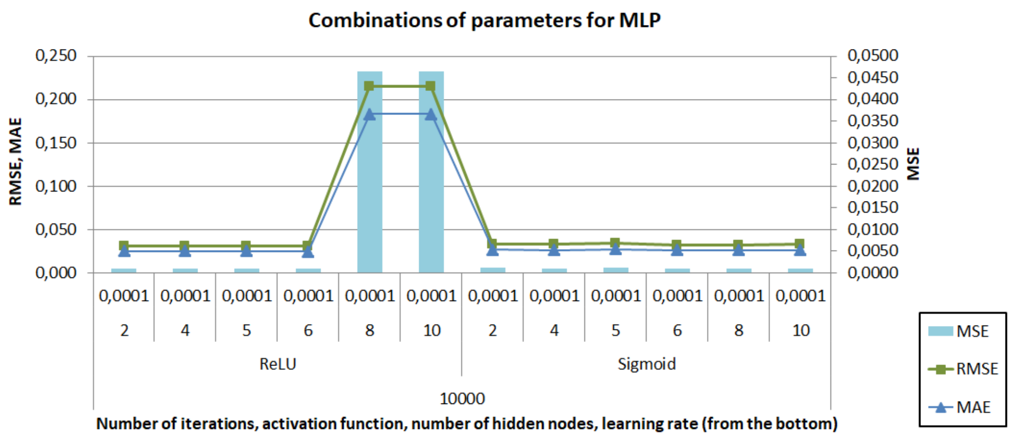


Figure 18. All combinations of parameters used to construct the MLP network for the nearly nonstationary time series forecasting and values of the performance metrics calculated on scaled validation data.

Activation function	Number of hidden nodes	Learning rate	Number of iterations	Validation data			Train data		
				MSE	RMSE	MAE	MSE	RMSE	MAE
ReLU	6	0.0001	10000	0.000942	0.030686	0.024773	0.000935	0.030570	0.024095
ReLU	2	0.0001	10000	0.000951	0.030843	0.024927	0.000941	0.030682	0.024137
ReLU	5	0.0001	10000	0.000956	0.030916	0.024972	0.000947	0.030775	0.024240
ReLU	4	0.0001	10000	0.000956	0.030921	0.024991	0.000947	0.030773	0.024196
Sigmoid	6	0.0001	10000	0.001039	0.032229	0.026069	0.001024	0.032006	0.025110
Sigmoid	8	0.0001	10000	0.001053	0.032448	0.026223	0.001042	0.032274	0.025325
Sigmoid	10	0.0001	10000	0.001065	0.032628	0.026386	0.001052	0.032442	0.025447
Sigmoid	4	0.0001	10000	0.001071	0.032727	0.026451	0.001059	0.032545	0.025530
Sigmoid	2	0.0001	10000	0.001096	0.033104	0.026782	0.001081	0.032874	0.025787
Sigmoid	5	0.0001	10000	0.001158	0.034037	0.027556	0.001140	0.033762	0.026469

Table 11. 10 combinations of parameters used to construct the MLP network for the nearly non-stationary time series forecasting giving smallest values of the performance metrics on the scaled validation data.

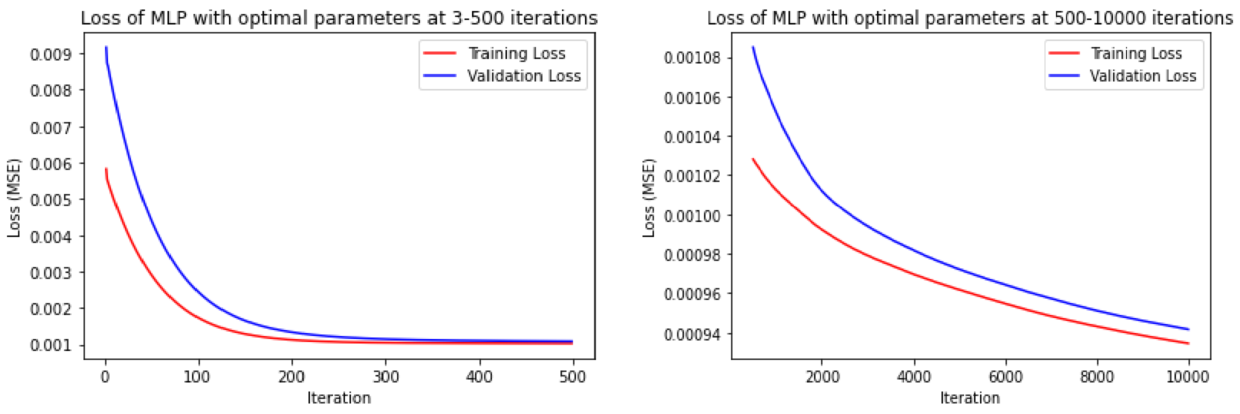


Figure 19. The training and validation loss (MSE) of the MLP with the optimal parameters after each of 3-500 iterations (on the left) and after the 500-10000 iterations (on the right). All calculation are made on the scaled data.

After the selection of optimal parameters there was important to verify if 10000 iterations does not cause overfitting. For this purpose the MSE function was used to calculate the errors of predictions in training and validation data sets after each iteration as it was done before. After the first two iterations, as might have been expected using randomly generated weights at the beginning of training the network, the training loss was quite big (1.687 and 0.738) while after the third iteration it decreased until 0.006. The validation loss also had a huge decrease after the second iteration (from 0.513 to 0.0078), then after the third iteration it slightly increased until 0.0091 and after that it started to steadily decrease. Figure 19 shows the both validation and training loss after each of 3-10000 iterations, by excluding first two iterations for the more clear demonstration without huge changes of values of loss functions. In general, validation and training loss functions steadily decrease and after the final iteration the difference between them is very small (about 0.000007), therefore, 10000 iterations does not cause overfitting and helps to reach quite small values of the performance metrics used in this experiment.

The performance of the constructed MLP model, used for the nearly nonstationary time series forecasting, was depicted visually in the Figures 20 and 21. It could be seen that all predictions are not very ideal, the small shift could be noticed and a little bit smaller values are predicted at the peaks of the data but, in principle, they look quite accurate.

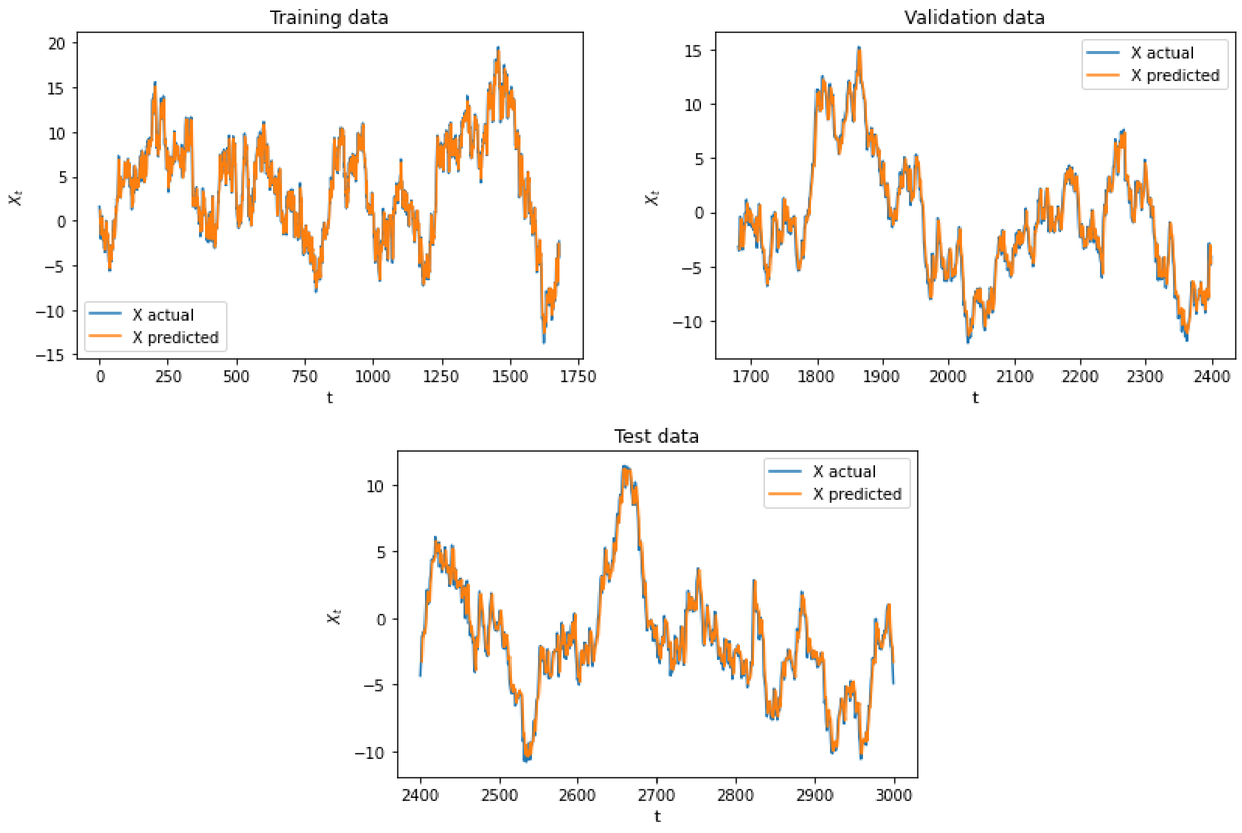


Figure 20. Actual and predicted nearly nonstationary AR(1) time series with $\beta = 0.99$ using MLP with optimal parameters (ReLU activation function, 6 hidden nodes, learning rate equals to 0.0001 and 10000 iterations).

3.1.2.2 Hyperparameters tuning of Recurrent Neural Network (RNN)

By using the same combinations of parameters for the construction of the RNN model for the nearly nonstationary time series forecasting it was noticed that some combinations, which showed quite well performance of the nonstationary time series forecasting, were outperformed by others. However, some combinations of network's parameters were not appropriate or gave quite small values of error functions for both nonstationary and nearly nonstationary time series forecasting. By comparing Figures 10 and 22 it could be noticed that the combination of two hidden nodes and a smaller learning rate (0.001) is not a good choice in an attempt to get smaller values of errors

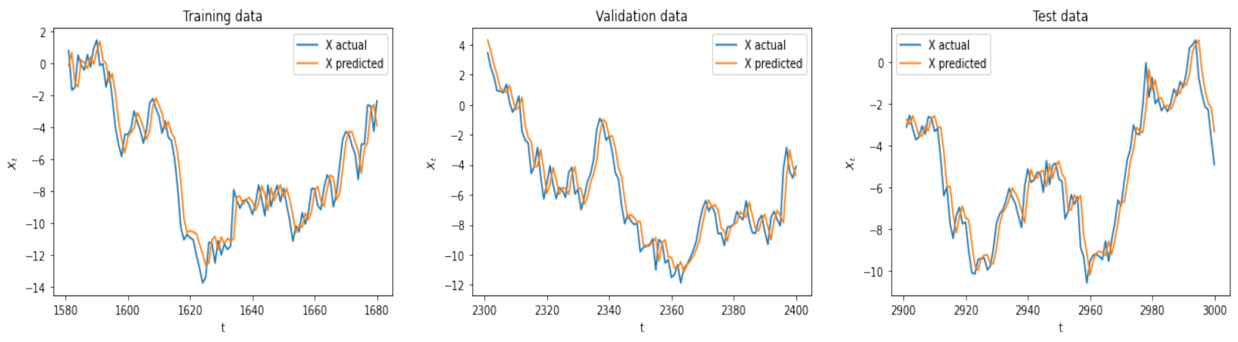


Figure 21. Last 100 actual and predicted values of training, validation and test data sets using MLP for the forecasting.

for both nearly nonstationary and nonstationary time series as well as sigmoid activation function with a smaller learning rate. It could also be seen that the increase of number of iterations does not cause very significant and clearly seen changes of values of performance metrics for the majority of combinations of parameters, especially when the ReLU activation function is used in the hidden layer.

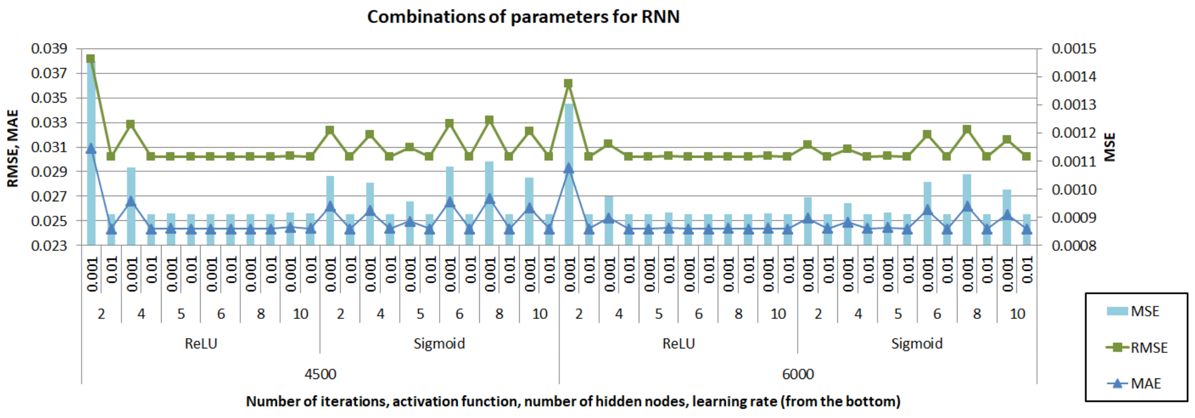


Figure 22. All combinations of parameters used to construct the RNN for the nearly nonstationary time series forecasting and values of performance metrics calculated on scaled validation data.

The Table 12 shows that, differently from the nonstationary time series forecasting using RNN, for the nearly nonstationary time series, applying sigmoid activation function in the hidden layer, smaller values of the performance metrics are obtained. Based on the value of RMSE, the most optimal combination of parameters consists of sigmoid function, 5 hidden nodes, 0.01 learning rate and 6000 iterations while the value of MAE is smallest when 6 hidden nodes and less iterations (4500) are used for the training of the network. The value of MSE after rounding is not very informative by trying to select a single combination because it is the same for all 7 first combinations in the Table 12. After considering obtained results, it was decided to choose the combination with sigmoid activation function, learning rates equals to 0.01, 6 hidden nodes and a smaller number of iterations (4500), due to the fact, that for this combination value of MAE is smallest and the value of RMSE is quite similar to the one which is obtained by using 5 hidden nodes and 6000 iterations.

By looking at the training and validation loss (MSE) functions, which are presented in the Figure 23, we could see that, at the beginning of the training of the network, both curves are fluctuating but after about 100 iterations they start to steadily decrease. This decrease continues till the end of iterating and after the last iteration values of training and validation loss functions are quite small and similar, the difference between them are only about 0.000000177. There was tried to slightly increase the number of iterations but it did not give very significant results in the performance of

Activation function	Number of hidden nodes	Learning rate	Number of iterations	Validation data			Train data		
				MSE	RMSE	MAE	MSE	RMSE	MAE
Sigmoid	5	0.01	6000	0.000911	0.030184	0.024353	0.000912	0.030191	0.023937
Sigmoid	6	0.01	4500	0.000911	0.030185	0.024345	0.000911	0.030188	0.023927
Sigmoid	5	0.01	4500	0.000911	0.030185	0.024357	0.000912	0.030195	0.023940
Sigmoid	6	0.01	6000	0.000911	0.030186	0.024347	0.000911	0.030184	0.023929
Sigmoid	8	0.01	6000	0.000911	0.030188	0.024347	0.000911	0.030183	0.023926
Sigmoid	10	0.01	6000	0.000911	0.030188	0.024347	0.000911	0.030181	0.023927
ReLU	6	0.001	4500	0.000911	0.030191	0.024356	0.000911	0.030177	0.023944
ReLU	10	0.01	6000	0.000912	0.030192	0.024349	0.000912	0.030191	0.023928
ReLU	6	0.001	6000	0.000912	0.030193	0.024357	0.000911	0.030176	0.023942
Sigmoid	10	0.01	4500	0.000912	0.030195	0.024350	0.000911	0.030189	0.023921

Table 12. 10 combinations of parameters used to construct the RNN network for the nearly nonstationary time series forecasting giving smallest values of the performance metrics on the scaled validation data.

predicting validation data and it was noticed that the validation loss starts to increase by using a slightly more iterations. Therefore, it was decided that 4500 iterations is an optimal choice and does not cause overfitting.

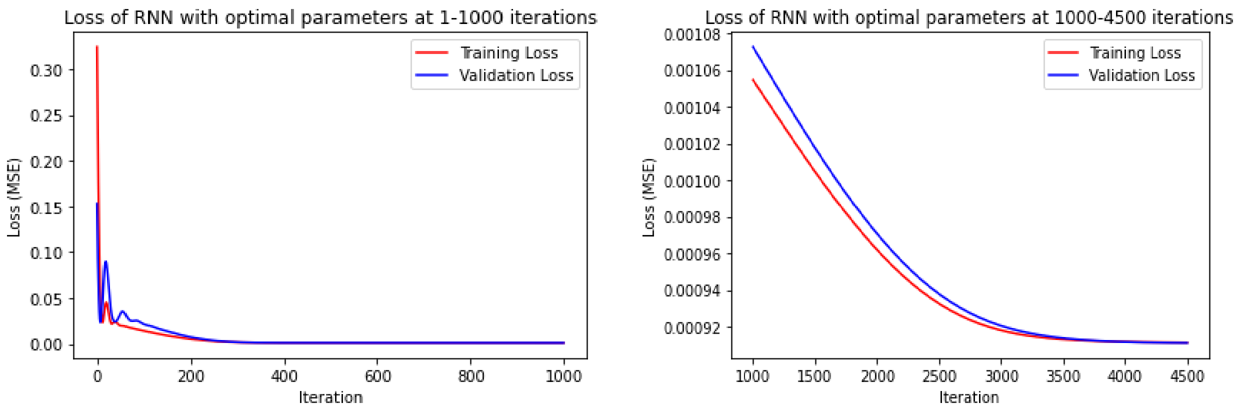


Figure 23. The training and validation loss (MSE) of the RNN with the optimal parameters after each of 1-1000 iterations (on the left) and after the 1000-4500 iterations (on the right). All calculation are made on the scaled data.

Real and predicted values of the nearly nonstationary AR(1) time series using RNN are visualized in the Figures 23 and 24. It looks like predictions using RNN are more accurate than using MLP and this models better deals with spikes in data. However, as in all previous figures, which show last 100 actual and predicted values, a small shift is noticeable.

3.1.2.3 Hyperparameters tuning of Support Vector Regression (SVR)

After applying the same SVR parameters for the nearly nonstationary time series forecasting as were used for the nonstationary time series, some similar tendencies have been found. By comparing Figures 14 and 26 it could be seen that by using RBF and sigmoid kernel functions, the values of metrics change quite similarly for both nearly nonstationary and nonstationary time series. There also exists a strong dependency between errors obtained and the value of kernel parameter γ, hence, for example, by choosing RBF kernel with smaller value of C and γ equals to 0.0001 as well as sigmoid kernel with bigger value of C and γ equals to 0.1 very inaccurate predictions are made. This is the reason why these huge jumps of values in Figures 14 and 26 exist. By taking linear kernel function for the nonstationary time series forecasting it was noticed that the best choice of the value of ϵ is 0.01 while for the nearly nonstationary time series, for the majority of combinations, the

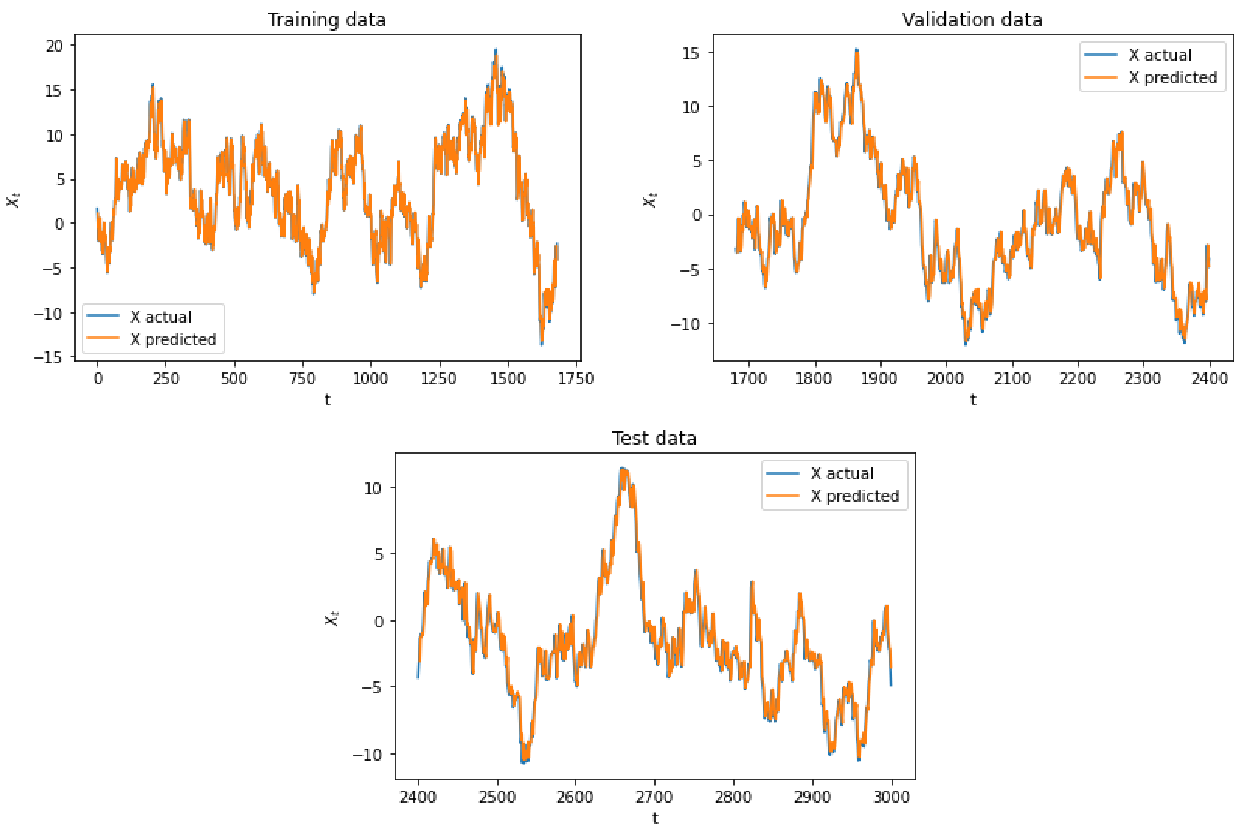


Figure 24. Actual and predicted nearly nonstationary AR(1) time series with $\beta = 0.99$ using RNN with optimal parameters (sigmoid activation function, 6 hidden nodes, learning rate equals to 0.01 and 4500 iterations).

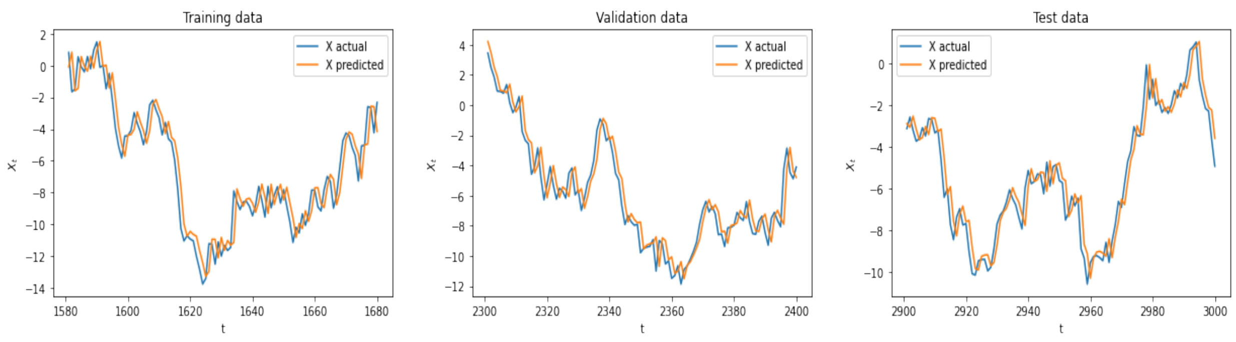


Figure 25. Last 100 actual and predicted values of training, validation and test data sets using RNN for the forecasting.

better results are obtained with the smaller value of ϵ, judging by the changes of MSE which are more clearly seen.

Although by looking at the graphical representations of the performance metrics, calculated on the validation data using different SVR parameters, we could see some similarities between nonstationary and nearly nonstationary time series, after comparing combinations, which gave the smallest values of metrics, some differences were found. From the Table 13 it could be noticed that the nearly nonstationary time series requires the smaller value of ϵ and the bigger value of C in an attempt to get more accurate predictions by comparing to the nonstationary time series. The values of MSE and RMSE are smallest when the sigmoid kernel is used with the values of C, ϵ and γ equal to 512, 0.00001 and 0.01 respectively. However, the most optimal combination of parameters, based on the value of MAE, consists of RBF kernel function with the values of C, ϵ and γ equal to

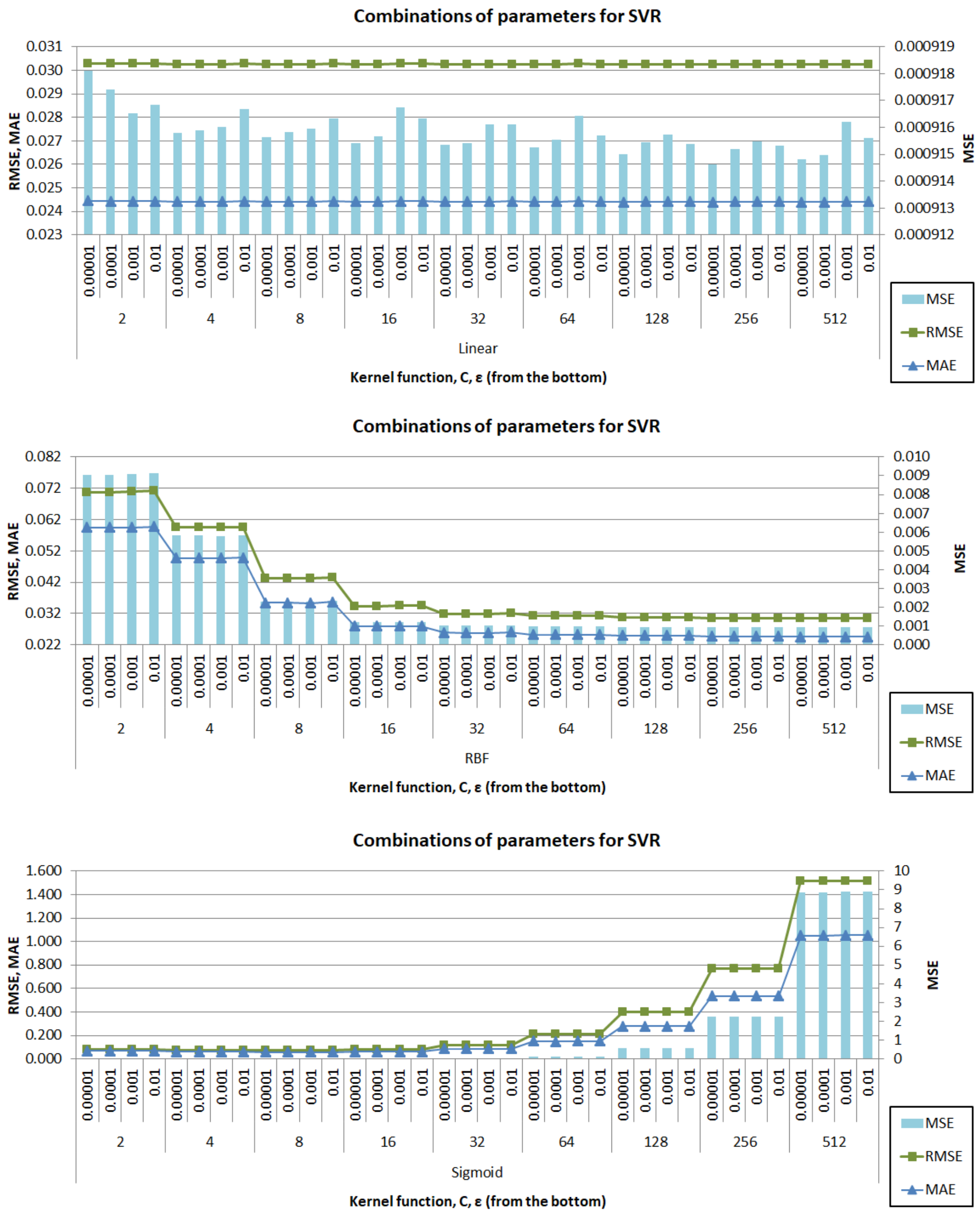


Figure 26. All combinations of three parameters used to construct the SVR for the nearly nonstationary time series forecasting and values of performance metrics calculated on scaled validation data.

256, 0.001 and 0.1 respectively. For the final decision whether the first or the second combination of parameters should be chosen as optimal, similarly as before (for the nonstationary time series forecasting), there was decided to calculate values of MASE. By using the first combination, the value of MASE was equal to 1.017411 while for the second combination it was slightly smaller (1.017306). Therefore, RBF kernel function with values of C, ϵ and γ equal to 256, 0.001 and 0.1 respectively were defined as optimal parameters.

For the purpose of demonstration of the real and predicted values of nearly nonstationary AR(1) time series using SVR, Figures 27 and 28 are included. It could be seen that predicted values are quite accurate but a small shift exists.

Kernel function	C	ϵ	γ	Validation data			Train data		
				MSE	RMSE	MAE	MSE	RMSE	MAE
Sigmoid	512	0.00001	0.01	0.0009142	0.0302361	0.0243865	0.0009133	0.0302217	0.0239153
RBF	256	0.001	0.1	0.0009145	0.0302406	0.0243840	0.0009138	0.0302294	0.0238875
Linear	256	0.00001		0.0009146	0.0302426	0.0243914	0.0009135	0.0302240	0.0239120
RBF	32	0.001	0.1	0.0009148	0.0302455	0.0243889	0.0009135	0.0302246	0.0238936
Linear	512	0.00001		0.0009148	0.0302458	0.0243943	0.0009133	0.0302202	0.0239113
RBF	64	0.0001	0.1	0.0009148	0.0302465	0.0243921	0.0009133	0.0302209	0.0238872
RBF	32	0.00001	0.1	0.0009149	0.0302479	0.0243947	0.0009138	0.0302294	0.0238899
Linear	512	0.0001		0.0009150	0.0302483	0.0243987	0.0009131	0.0302182	0.0239113
Linear	128	0.00001		0.0009150	0.0302488	0.0243973	0.0009139	0.0302300	0.0239129
Sigmoid	512	0.0001	0.01	0.0009150	0.0302492	0.0244002	0.0009133	0.0302212	0.0239155

Table 13. 10 combinations of parameters used to construct the SVR for the nearly nonstationary time series forecasting giving smallest values of the performance metrics on the scaled validation data.

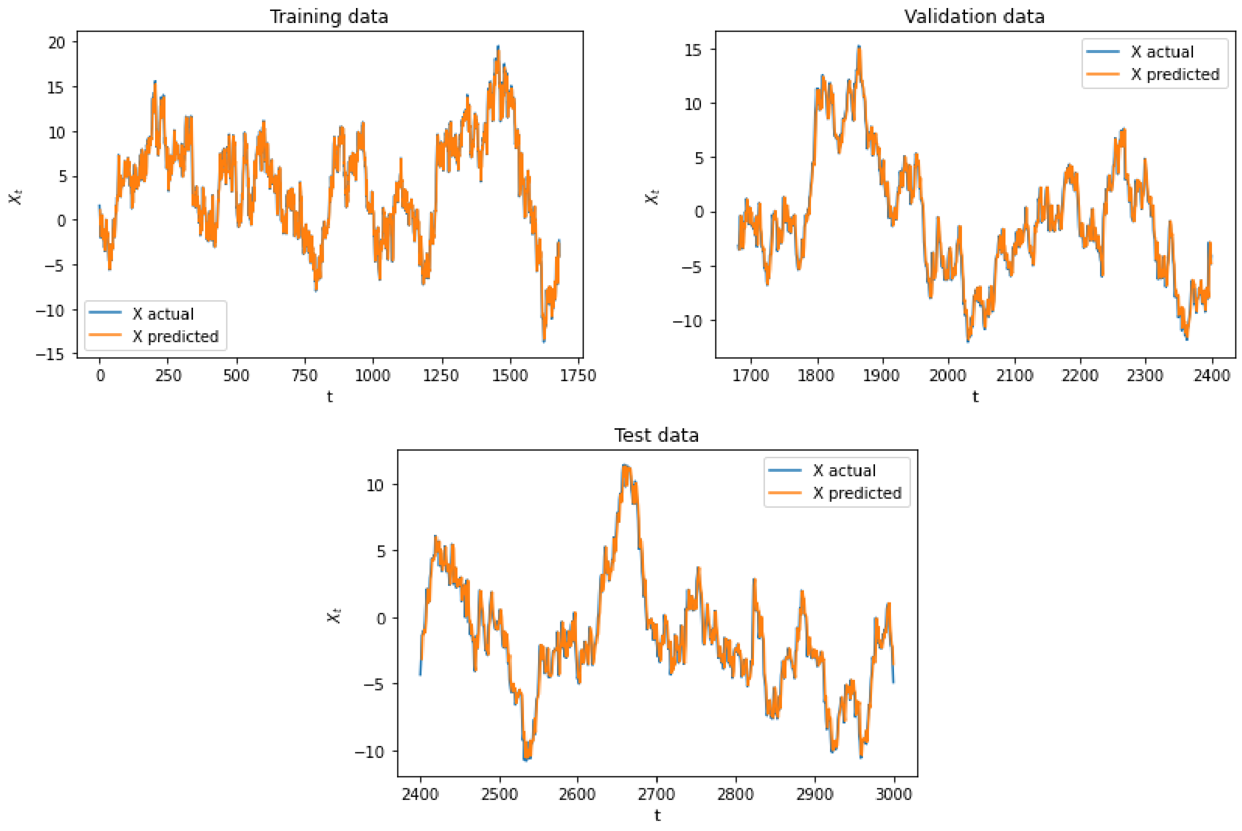


Figure 27. Actual and predicted nearly nonstationary AR(1) time series with $\beta = 0.99$ using SVR with optimal parameters (RBF kernel function with values of C, ϵ and γ equal to 256, 0.001 and 0.1 respectively).

3.1.2.4 Application of ARIMA

From the theory it is known, that when the absolute value of the coefficient β of the AR(1) time series is smaller than 1, then the time series is assumed to be stationary. The stationary AR(1) is an equivalent to the ARIMA(1,0,0), therefore the future values of this time series could be predicted based on the previous observations. In our case, the AR(1) time series was simulated by taking the value of β equals to 0.99 which is smaller than 1, thus the predictions of this time series could simply be found by using ARIMA(1,0,0) model. Similarly as for the nonstationary time series forecasting, the model was fitted on the first 2400 data points and then for the each observation in the test data

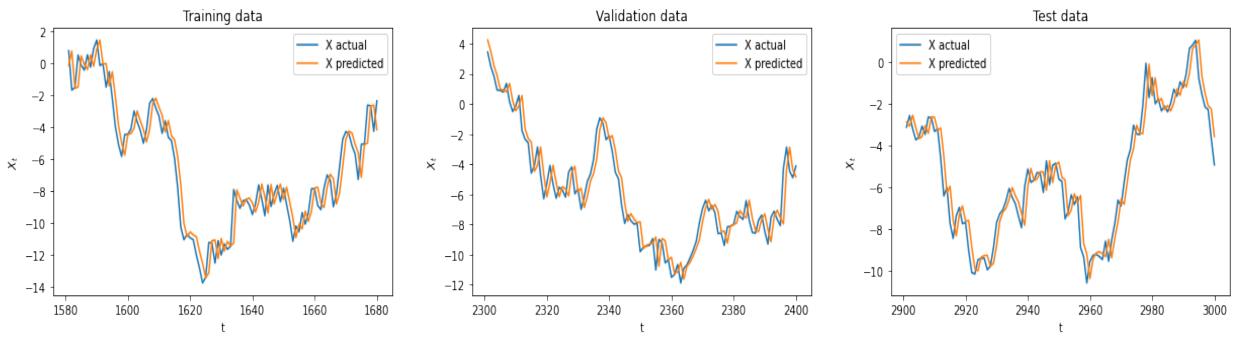


Figure 28. Last 100 actual and predicted values of training, validation and test data sets using SVR for the forecasting.

set the predictions were generated using ARIMA(1,0,0). Real and predicted values of the test data set are shown in the Figure 29. It looks like from the visual perspective that the traditional statistical model ARIMA is an appropriate choice for the linear nearly nonstationary time series and using this model quite accurate predictions could be generated.

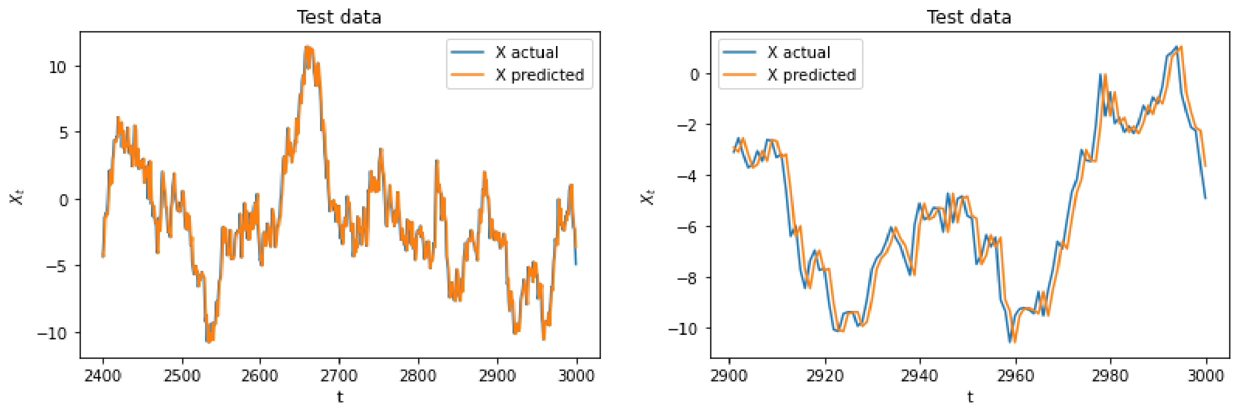


Figure 29. Actual and predicted test data of nearly nonstationary AR(1) time series with $\beta = 0.99$ using ARIMA(1,0,0) (on the left) by extracting additionally last 100 values (on the right).

In addition, it was decided to apply automatic ARIMA which is able to use unit root tests and some information criterion and itself generates the optimal values of parameters p, d and q that would be suitable for the data to provide better forecasting (see more in [32]). The suggestion of automatic ARIMA was to use ARIMA(0,1,0) which shows that, even though the value of AR(1) coefficient β is smaller than 1 and the time series are treated as stationary, it actually behaves more likely to nonstationary time series and are suggested to be differenced. From the visual perspective the obtained predictions using ARIMA(0,1,0) look very similarly as using ARIMA(1,0,0), therefore the visual representation is not included here and the accuracy of forecasting between these two models is compared in the following section.

3.1.2.5 Comparison of the performance of models of the single AR(1) realization forecasting

Final evaluations of the forecasting performance of each model obtained by using four metrics are presented in the Table 14. It could be seen that, based on the values of MSE, RMSE and MASE, the most accurate predictions are obtained by using ARIMA(1,0,0) which is an equivalent to the analyzed AR(1) time series. However, SVR shows also quite well forecasting performance and, based on the values of MAE, it even slightly outperforms the ARIMA(1,0,0). It is interesting that the model suggested by the automatic ARIMA performs worse than original ARIMA(1,0,0), by showing that nearly nonstationary time series should not be predicted as the nonstationary time series. These results could be compared to those in the research in [8] where also nearly nonstationary AR(1) were predicted. Authors of this article found that for the simulated AR(1) processes with $1 \geqslant \beta \geqslant 0.9$ it is suggested to use differencing to obtain better forecasting performance, however, in this experiment we could see that more accurate predictions could be made without differencing of time series. The reason of these different results could be that the size of samples significantly differs: in the research in [8] there are 100 generated data points while in this experiment the length of the time series is equal to 3000. Based on the conclusions of [17] it was noticed that for the nearly nonstationary time series with a larger sample size it is more preferable to use the true models of processes, as in our case the ARIMA(0,1,0) for the AR(1) forecasting, therefore these findings can be confirmed in this experiment. Based on the value of MAE the ARIMA(0,1,0) model is even the worst choice for the nearly nonstationary time series forcasting, while the values of MSE, RMSE and MASE show that it is better to predict values of the nearly nonstationary time series using the simple statistical model, which is appropriate for the nonsationary time series forecasting, instead of the ML model MLP.

From the visual perspective, similarly as in nonstationary time series forecasting, the shift between real and predicted values were noticed by using all models. The reason could be that, as it was noticed in the literature review, the nearly nonstationary AR(1) time series often has similar behaviour to the nonstationary time series which are the random walk processes and the suggestion of automatic ARIMA also confirms that there are some similarities between these processes. After many attempts to find the optimal parameters of the models the best obtained predictions were always quite close to the values of the previous observation. However, based on the values of MASE we could see that all models under consideration predict slightly more accurately than the naïve model in-sample.

	MSE	**RMSE**	**MAE**	**MASE**
MLP	0.922818	0.960634	0.770478	0.965926
RNN	0.911371	0.954657	0.768276	0.963165
SVR	0.909887	0.953880	0.767394	0.962060
ARIMA(1,0,0)	0.909636	0.953749	0.768066	0.957243
ARIMA(0,1,0)	0.915987	0.957072	0.772890	0.963255

Table 14. Final values of metrics used to evaluate the forecasting performance of models on the test data set of the single nearly nonstationary AR(1) realization.

3.1.2.6 Comparison of the performance of models of the 100 AR(1) realizations forecasting

After applying all models with the defined optimal parameters for the 100 nearly nonstationary AR(1) realizations forecasting, the final evaluations of the forecasting performance were made and presented in the Table 15. It was noticed that the best forecasting performance could be reached by using the original AR(1) equivalent - ARIMA(1,0,0) model. However, RNN and SVR could obtain also quite accurate predictions and the ability of RNN to remember previous calculations and make the training process sequential helps to reach quite well results, even slightly better than SVR could achieve which was one of the best models for the single realization of the nearly nonstationary AR(1) forecasting. However, based on the average values of metrics, all ML models are outperformed by traditional statistical ARIMA models, even by the ARIMA(0,1,0) which was outperformed by SVR and RNN when the single realization of the nearly nonstationary AR(1) was predicted. The differences between the average errors, computed using ARIMA(1,0,0) and ARIMA(0,1,0), are not very significant, therefore overdifferencing of the nearly nonstationary time series does not improve our predictions but it also does not cause high inefficiency in forecasting.

Another remark of this experiment is that all ML models by using the same parameters (which, possibly, are not the most optimal for all simulated time series) for the more realizations of the nearly nonstationary time series forecasting could manage significantly better than predicting more different nonstationary time series realizations. Therefore it could be assumed that the accuracy of predictions of the nearly nonstationary time series is less dependent on the model parameters, probably, due to the fact that those time series do not have so many radical changes between each other or very unstable mean and variance as nonstationary time series have.

	MSE			**RMSE**			**MAE**			**MASE**		
	MIN	**MAX**	**AVG**	**MIN**	**MAX**	**AVG**	**MIN**	**MAX**	**AVG**	**MIN**	**MAX**	**AVG**
MLP	0.9129	4.3813	1.1630	0.9554	2.0931	1.0664	0.7601	1.3359	0.8431	0.9345	1.6562	1.0532
RNN	0.8964	1.3565	1.0338	0.9468	1.1647	1.0159	0.7465	0.9340	0.8104	0.9131	1.1888	1.0125
SVR	0.8955	1.8217	1.0392	0.9463	1.3497	1.0181	0.7453	1.0108	0.8114	0.9116	1.2395	1.0137
ARIMA(1,0,0)	0.8937	1.1726	1.0091	0.9454	1.0829	1.0041	0.7465	0.8713	0.8011	0.9211	1.0915	1.0004
ARIMA(0,1,0)	0.8950	1.1762	1.0135	0.9460	1.0845	1.0063	0.7459	0.8688	0.8029	0.9236	1.0909	1.0027

Table 15. Minimum, maximum and average values of metrics used to evaluate the forecasting performance of models on the test data set of the 100 nearly nonstationary AR(1) realizations.

3.1.2.7 The effect of the changing coefficient β of AR(1) for the performance of models

For the purpose of making more general conclusions about the nearly nonstationary time series forecasting it was decided to apply already constructed models for the 100 simulated AR(1) time series realizations with different values of parameter β. The aim of this part of the experiment was to check how the forecasting performance of models changes when the time series approaches stationarity, therefore all models with the same hyperparameters, which have been assumed as optimal for the single nearly nonstationary AR(1) forecasting, were used to more clearly see the effect of changing the value of β for the accuracy of predictions. Two values of parameter β equal to 0.97 and 0.95 were chosen and, similarly as before, four metrics were used (MSE, RMSE, MAE and MASE) for the forecasting performance of each model evaluations. Minimum, maximum and average values of the forecasting performance metrics, calculated on the test data sets of each of simulated realizations of AR(1), are presented in the Table 16.

β	**Model**	**MSE**			**RMSE**			**MAE**			**MASE**		
		MIN	**MAX**	**AVG**	**MIN**	**MAX**	**AVG**	**MIN**	**MAX**	**AVG**	**MIN**	**MAX**	**AVG**
	MLP	0.8955	1.4330	1.0318	0.9463	1.1971	1.0150	0.7485	0.9222	0.8090	0.9098	1.1580	1.0055
	RNN	0.8979	1.1997	1.0146	0.9476	1.0953	1.0068	0.7476	0.8853	0.8032	0.9063	1.1117	0.9983
0.97	SVR	0.8978	1.3048	1.0204	0.9475	1.1423	1.0096	0.7453	0.8906	0.8052	0.9036	1.1125	1.0008
	ARIMA(1,0,0)	0.8939	1.1715	1.0093	0.9455	1.0824	1.0042	0.7462	0.8706	0.8011	0.9135	1.0873	0.9954
	ARIMA(0,1,0)	0.9038	1.1896	1.0239	0.9507	1.0907	1.0114	0.7480	0.8724	0.8071	0.9186	1.0968	1.0028
	MLP	0.8934	1.1958	1.0168	0.9452	1.0935	1.0079	0.7462	0.8862	0.8041	0.9005	1.1059	0.9943
	RNN	0.8999	1.1787	1.0126	0.9486	1.0857	1.0058	0.7483	0.8749	0.8025	0.9009	1.0970	0.9923
0.95	SVR	0.8969	1.1871	1.0161	0.9470	1.0895	1.0076	0.7458	0.8787	0.8040	0.8979	1.0966	0.9942
	ARIMA(1,0,0)	0.8938	1.1710	1.0093	0.9454	1.0821	1.0042	0.7460	0.8705	0.8012	0.9079	1.0818	0.9903
	ARIMA(0,1,0)	0.9121	1.2042	1.0344	0.9550	1.0974	1.0166	0.7521	0.8765	0.8115	0.9166	1.0987	1.0031

Table 16. Minimum, maximum and average values of metrics used to evaluate the forecasting performance of models on the test data set of the 100 AR(1) with β equals to 0.97 and 0.95 realizations.

As might have been expected, the true/equivalent model (ARIMA(1,0,0)) of the AR(1) time series shows the best forecasting performance. From the comparison of models ARIMA(1,0,0) and ARIMA(0,1,0) it could be seen that overdifferencing of the time series does not help to improve the performance of forecasting and these processes with β equals to 0.97 or, especially, to 0.95 are more close to stationarity than to nonstationarity. It could be noticed that average values of MSE, RMSE and MAE are the same or very similar when the ARIMA(1,0,0) is used for the AR(1) with β equals to 0.97 and 0.95 forecasting, but, when we use ARIMA(0,1,0), average values of errors increase when the value of β decreases. Also it could be noted that even though ARIMA(0,1,0) outperformed ML models for the 100 AR(1) with $\beta = 0.99$ realizations forecasting, by reducing the value of β to 0.97 it outperforms only the MLP model and with $\beta = 0.95$ ARIMA(0,1,0) is even the worst model giving the largest forecasting errors. These results confirm the idea which was mentioned in the literature review, that it is very important to test the stationarity of the time series before using forecasting models and overdifferencing can cause some inefficiencies in forecasting.

Even though the ARIMA(1,0,0) outperforms ML models, they also are able to make quite accurate predictions for the AR(1) time series with β equal to 0.97 or 0.95, even if selected hyper-parameters are not the most optimal for all of those time series. Possibly, by predicting a specific time series and choosing more optimal parameters for this case, the accuracy of predictions obtained using ML models would be even more accurate. Additionally, it could be noticed that with a smaller value of β AR(1) time series become more stable and easier to be predicted for all of analyzed ML models based on the fact that, when the value of β decreases from 0.97 to 0.95, the values of all forecasting performance metrics also decreases that shows smaller forecasting errors obtained. Similarly as in the comparison of models used to make the predictions for the 100 AR(1) time series with $\beta = 0.99$, by taking smaller values of β the best forecasting performance shows RNN model compared to other ML models as well as MLP performs worst. However, by taking into account all these results and some findings in the related works, where the ability of ML models to deal with strongly nonstationary and nonlinear time series has been proved as well as some difficulties in handling with linear parts of the time series have been noticed by using ML models, this simulation study could also confirm that traditional statistical models (without differencing) should be the more appropriate choice for the time series which approach stationarity, especially if they are linear (as in the analyzed case).

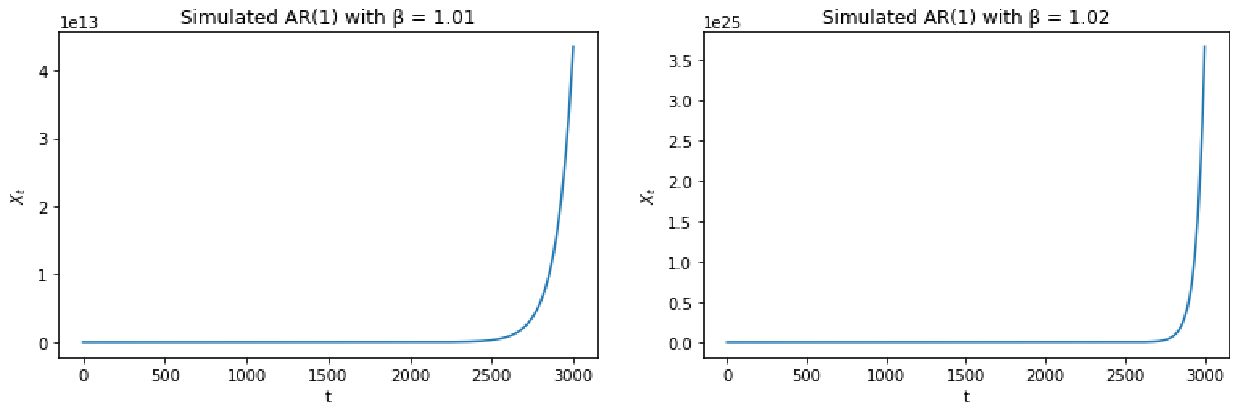


Figure 30. Simulated AR(1) time series with $\beta = 1.01$ (on the left) and $\beta = 1.02$ (on the right).

Additionally, it was decided to apply the same constructed models for the explosive AR(1) time series with β value equal to 1.01 and 1.02 for the purpose of testing if these models are also able to deal with blowing up in finite sample. First of all, the single simulated AR(1) with $\beta = 1.01$ and another with $\beta = 1.02$ were tested. Both time series are visually depicted in the Figure 30. By keeping the same data splitting into training, validation and test data sets we could see that there is a huge explosion in the test data set while in other data subsets the changes of values in time are not so big, therefore it was interesting to check if constructed ML models will be able to make quite accurate predictions for this explosion after learning from the more stable part of time series.

However, from the Figure 31 we could see that ARIMA models make quite accurate predictions for the test data of the explosive AR(1) time series and significantly outperforms constructed ML models. It is interesting that MLP, which was outperformed by RNN and SVR in all previous cases, is able to deal with the high explosion in time series, while other two ML models are completely incapable to predict these sharp rises of simulated AR(1). Due to the fact that values of the time

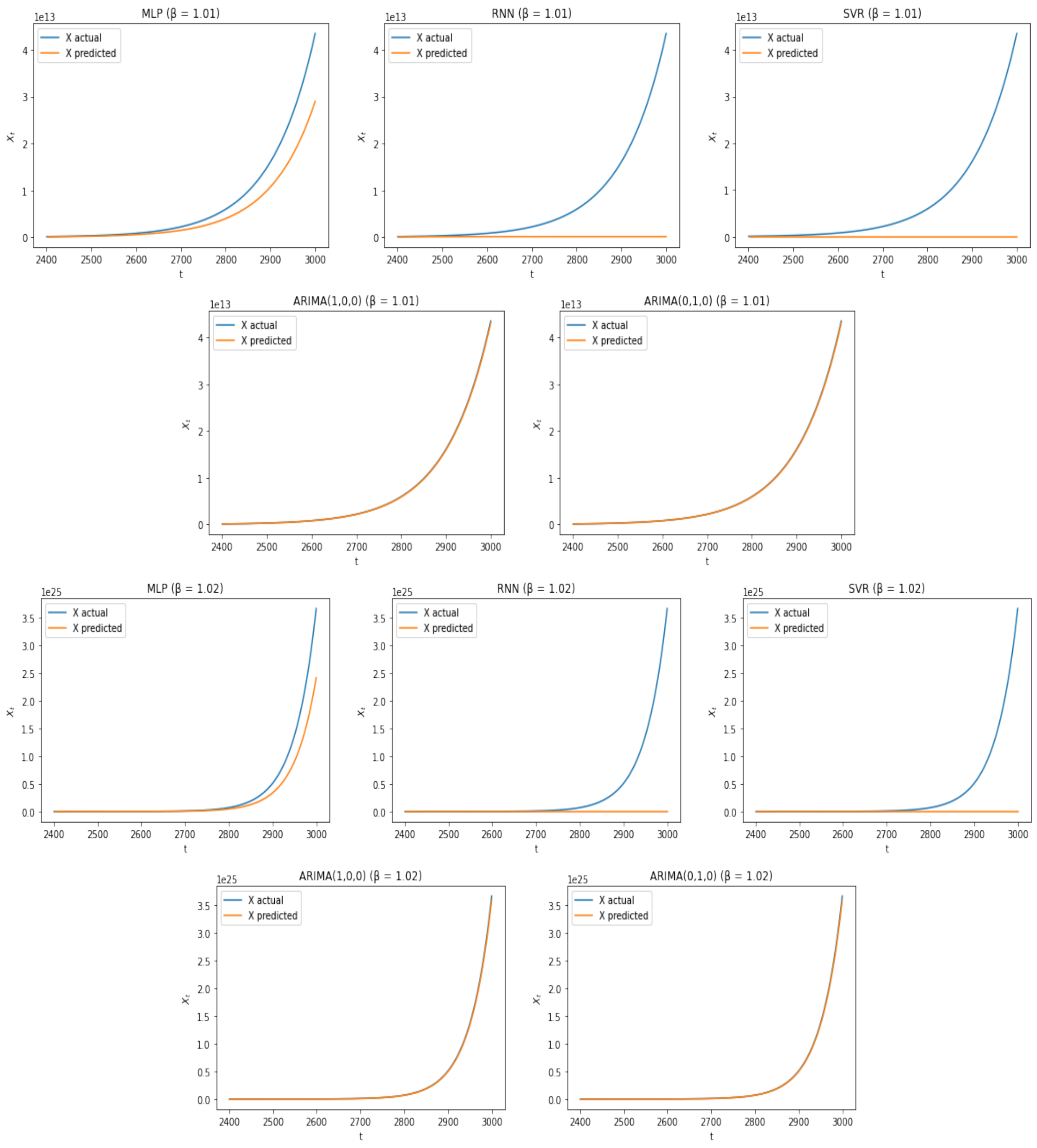
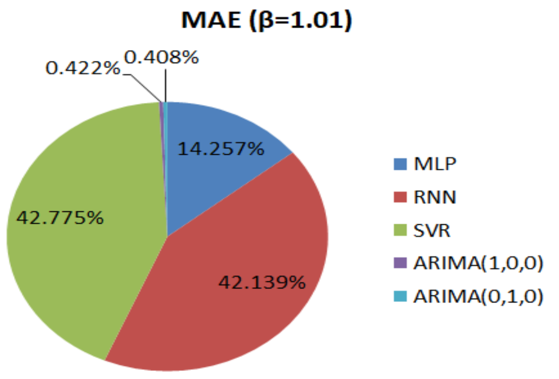


Figure 31. Actual and predicted values of AR(1) with β equals to 1.01 and 1.02 using all constructed models for the forecasting.

series as well as values of the performance metrics are very large and, based on all metrics, models are ranked in the same order by their accuracy, the Figure 32 is included to show the percentage values of the single metric (MAE) of each model. It could be seen that RNN and SVR obtain equally large errors and compared to MLP they are three times larger, while the errors obtained using ARIMA models are relatively small. In addition, all models were applied for more realizations of these explosive AR(1) time series, however, all models showed similar forecasting performance for all of the realizations.

Therefore, it could be concluded that constructed ML models, unfortunately, did not succeed in making accurate predictions for the AR(1) time series with value of β larger than 1, even though they were able to quite well forecast many different realizations of the AR(1) with β smaller that 1. These experimental results could indicate that the forecasting of the explosive AR(1) time series is



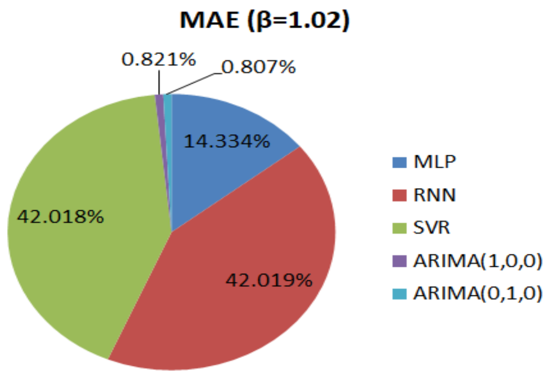


Figure 32. Simulated AR(1) time series with $\beta = 1.01$ (on the left) and $\beta = 1.02$ (on the right).

a special case which has to be analyzed separately in the further researches. The ability of MLP to capture the explosion of time series allows for us to assume that the selection of the more appropriate hyperparameters of the ML models could help to improve their forecasting performance and, perhaps, to make quite accurate predictions.

3.2 The performance of models of the real world data forecasting

One of the main interests of this work is to test the performance of ML models of the time series, with a high degree of persistence, forecasting. The random walk, analyzed in the simulation study, is one of the best known examples of a time series with a high degree of persistence as well as the nearly nonstationary AR(1) time series with a value of the parameter β close to 1, which also has a similar behaviour to random walk. It is known that the time series of financial market usually follow a random walk, therefore it was decided that such time series application could be useful in this experiment. The data of the Offset Market Exchange Gross Index of Vilnius (OMX Vilnius GI) were used (found in the Nasdaq official page). This the local all-share index is one of the best known indexes in the Lithuania which includes all the stocks on the Main and Secondary lists of the Vilnius market. OMX Vilnius GI shows the gross return of the stocks it includes.

Daily values of the index from 2000-01-01 to 2020-12-18 have been used in the experiment. Before applying models for the forecasting of this real world time series, it was important to eliminate duplicates and add missing dates by replacing missing values into average values obtained using the rolling mean which is better known as a simple moving average (SMA). SMA uses a sliding window to take the average of values over a set number of time periods. Due to the facts that the daily data are analyzed and the maximum period, containing missing values, was 6 days, the size of the window was set to 7 by calculating the SMA in each of 7 days period and using computed values instead of missing ones. Completely prepared data are visualized in the Figure 33.

Based on the fact that the time series of financial market follow a random walk, it was assumed that the analyzed time series could be modeled by the AR(1) model. The Yule-Walker method was used to estimate the AR(1) parameter β and the estimated value was equal to 0.99948474. Therefore, based on the assumption which has been made, it could be said that the analyzed real world time series is nearly nonstationary, because the estimated value of β is less than one but very close to unity.

The preparation of data for the forecasting was the same as in the simulation study (see more in the beginning of the Section 3.1), time series was divided into training, validation ant test data sets by keeping the same proportions and all values were scaled using the Min-Max scaling. The training data set consisted of the fist 4288 index values from 2000-01-01 to 2011-09-27, the validation data set contained further 1838 observations from 2011-09-28 to 2016-10-08 and for the testing the rest 1532 values were used from 2016-10-09 to 2020-12-18. After that, both neural network

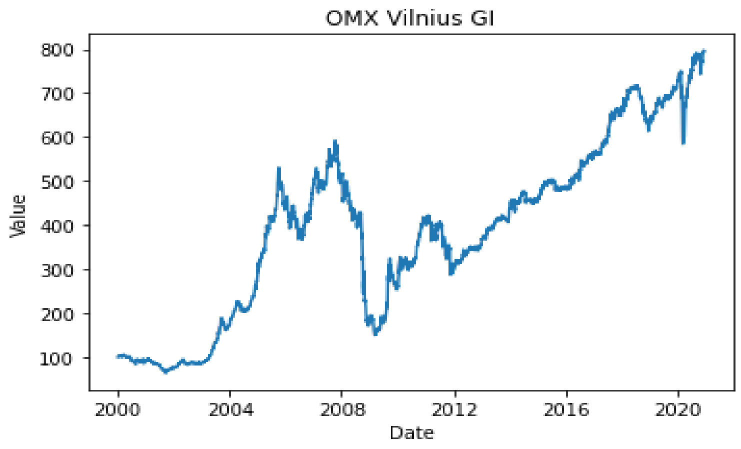


Figure 33. Prepared data of the Offset Market Exchange Gross Index of Vilnius (OMX Vilnius GI) used in the experiment.

models with the single hidden layer (MLP and RNN) and SVR model have been applied for the analyzed time series by trying to find optimal hyperparameters of models. Many different combinations of parameters have been tested and it was noticed that all ML models with significantly different parameters, compared to those which have been tested in the simulation study, obtain very inaccurate predictions. The optimal combination of parameters was selected based on the smallest values of the forecasting performance metrics, calculated on the validation data. It was found that MLP with ReLU activation function in the hidden layer, 8 hidden nodes, size of the learning rate equals to 0.00001 and 80000 iterations shows the best forecasting performance. Using RNN most accurate predictions were obtained with ReLU activation function, 6 hidden nodes, 0.01 learning rate and 6000 iterations. The most optimal combination of parameters for the SVR was RBF kernel function with values of values of C, ϵ and γ equal to 1024, 0.0001 and 0.1 respectively. All ML models with the selected optimal parameters and two statistical models (ARIMA(1,0,0) and ARIMA(0,1,0)) were applied for the time series of the OMX Vilnius GI forecasting. Actual and predicted values of this time series using each of five analyzed models are visually depicted in the Figures 34, 35, 36 and 37. By looking at these Figures, it looks like all models made quite accurate predictions, therefore, for the more clear demonstration, the Figures 38, 39, 40 and 41 are included, which show last 100 actual and predicted values of analyzed time series in each of the data subsets (training, validation and test). It could be noticed that all models predict not very ideally but quite accurate. From the visual perspective it looks like there are no significant differences in the predictions which are obtained using ARIMA(1,0,0) and ARIMA(0,1,0). RNN also shows quite good forecasting performance, while MLP and SVR are not able to deal with all spikes in data. It is interesting, that SVR, which in the majority of previous examples showed quite well forecasting performance and often outperformed neural network models, especially MLP, in this case performs worst, at least by looking at the predictions of the testing data. For the final comparison of the forecasting performance of models Table 17 is included in which the values of metrics, used to evaluate the accuracy of predictions on the test data set, are presented.

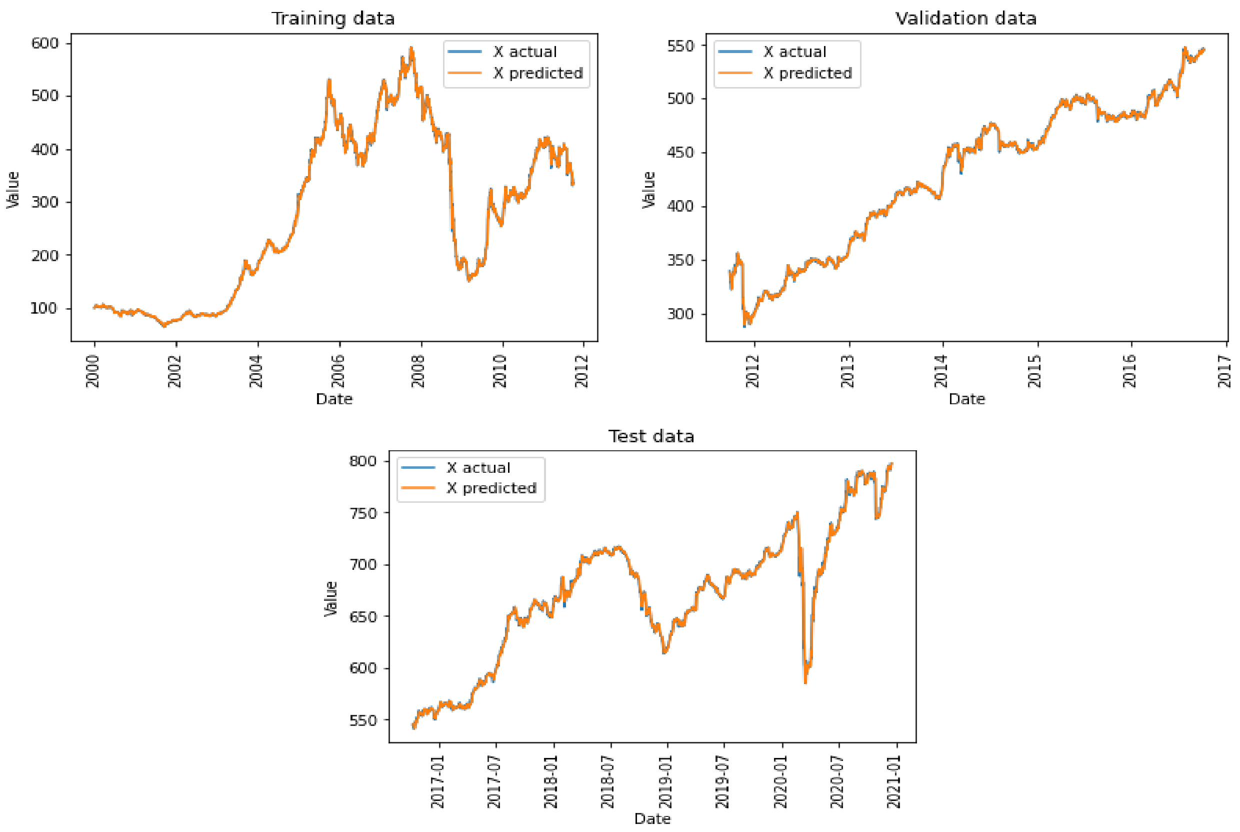


Figure 34. Actual and predicted values of the OMX Vilnius GI using MLP with optimal parameters (ReLU activation function, 8 hidden nodes, learning rate equals to 0.00001 and 80000 iterations).

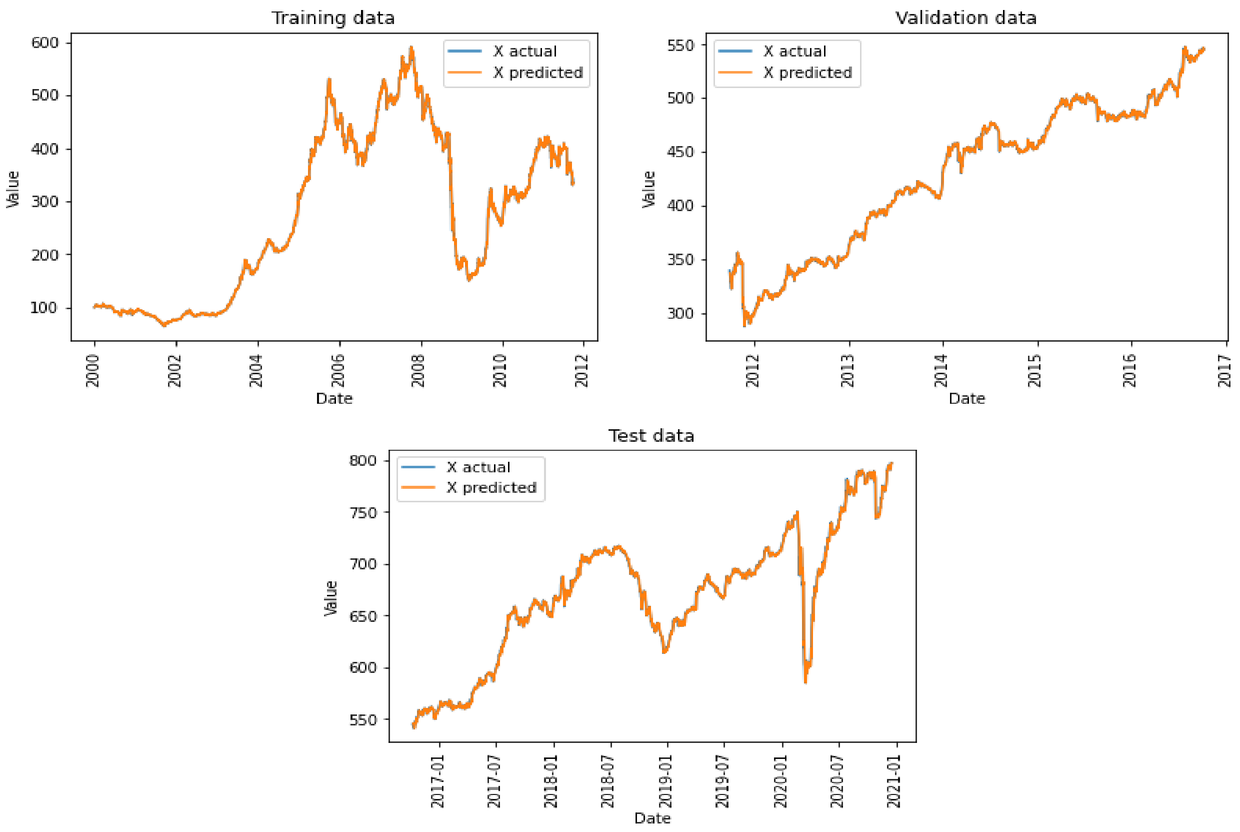


Figure 35. Actual and predicted values of the OMX Vilnius GI using RNN with optimal parameters (ReLU activation function, 6 hidden nodes, learning rate equals to 0.01 and 6000 iterations).

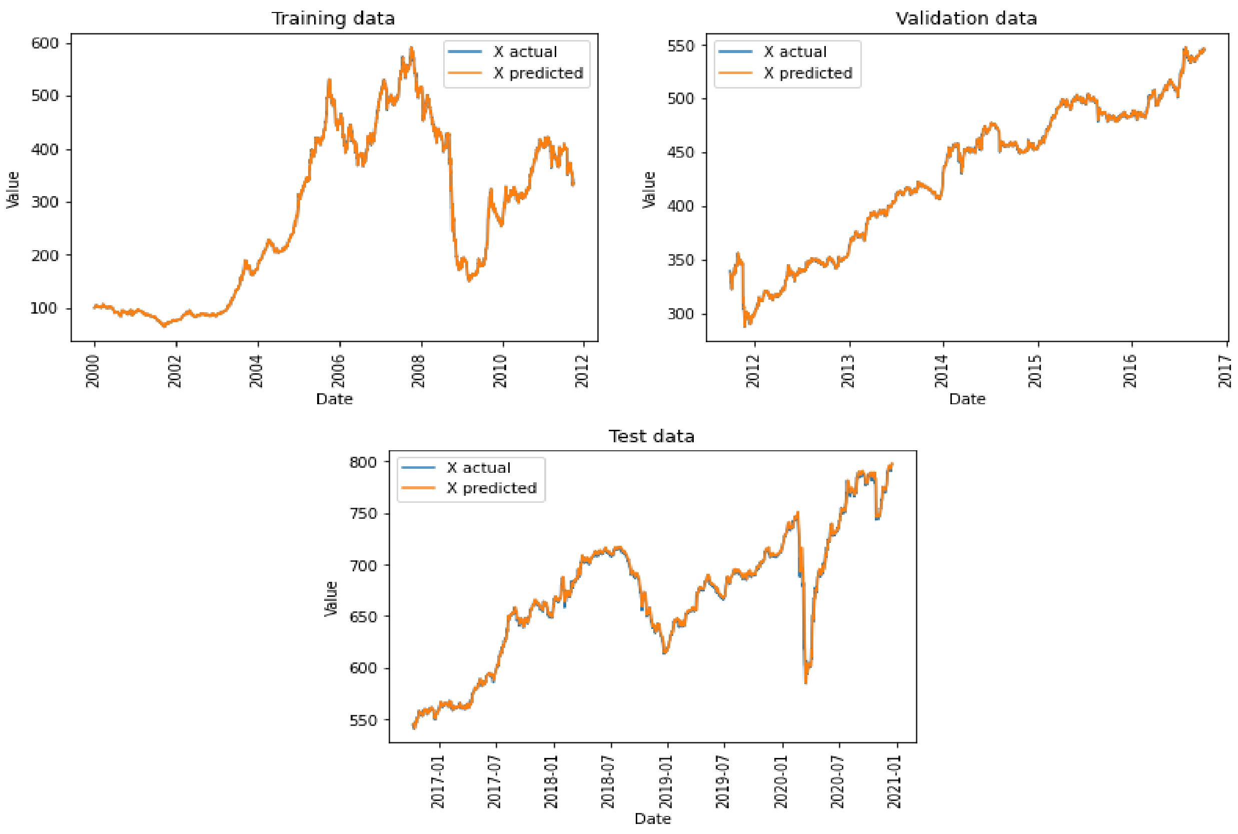


Figure 36. Actual and predicted values of the OMX Vilnius GI using SVR with optimal parameters (RBF kernel function with values of C, ϵ and γ equal to 1024, 0.0001 and 0.1 respectively).

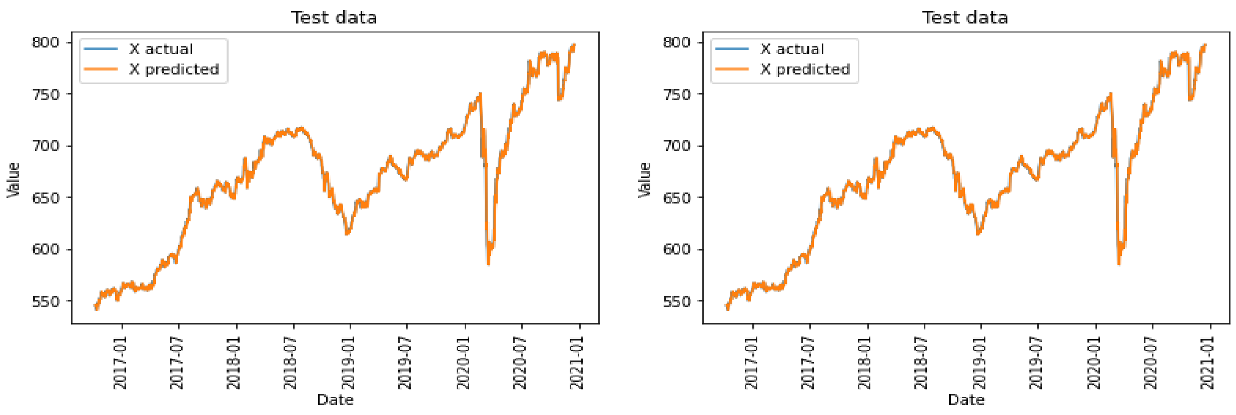


Figure 37. Actual and predicted test data of the OMX Vilnius GI using ARIMA(1,0,0) (on the left) and ARIMA(0,1,0) (on the right).

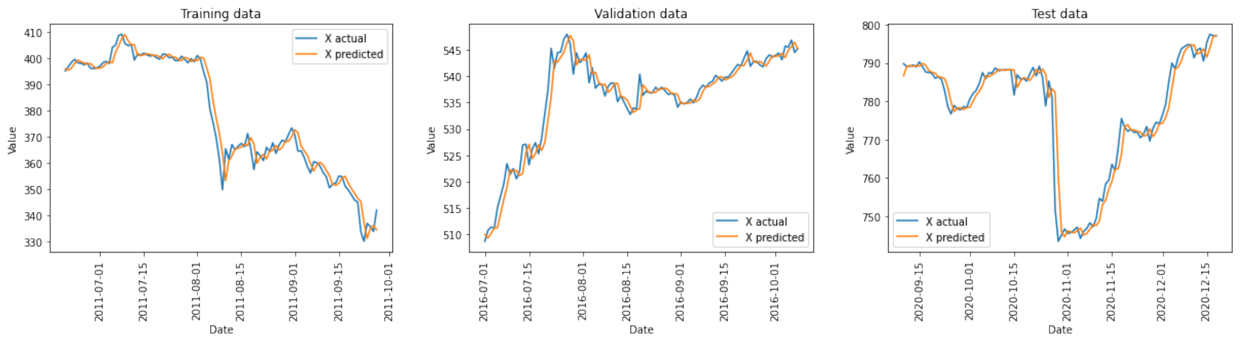


Figure 38. Last 100 actual and predicted values of training, validation and test data sets of the OMX Vilnius GI using MLP for the forecasting.

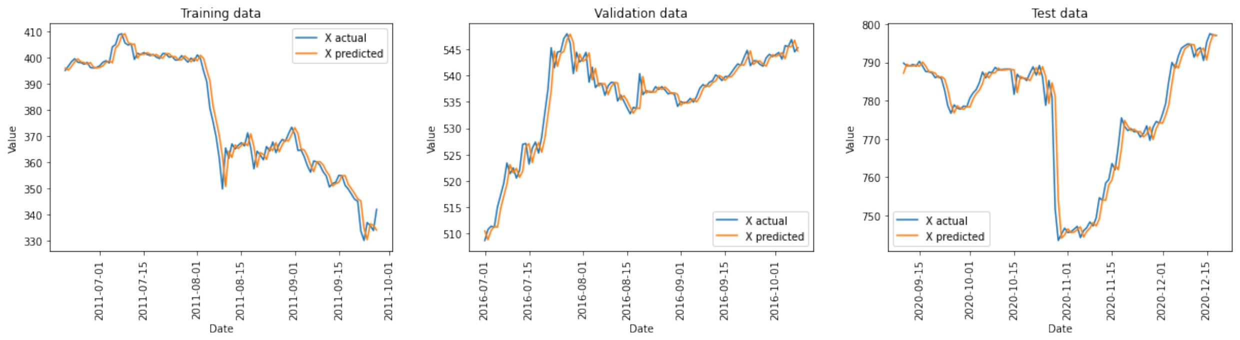


Figure 39. Last 100 actual and predicted values of training, validation and test data sets of the OMX Vilnius GI using RNN for the forecasting.

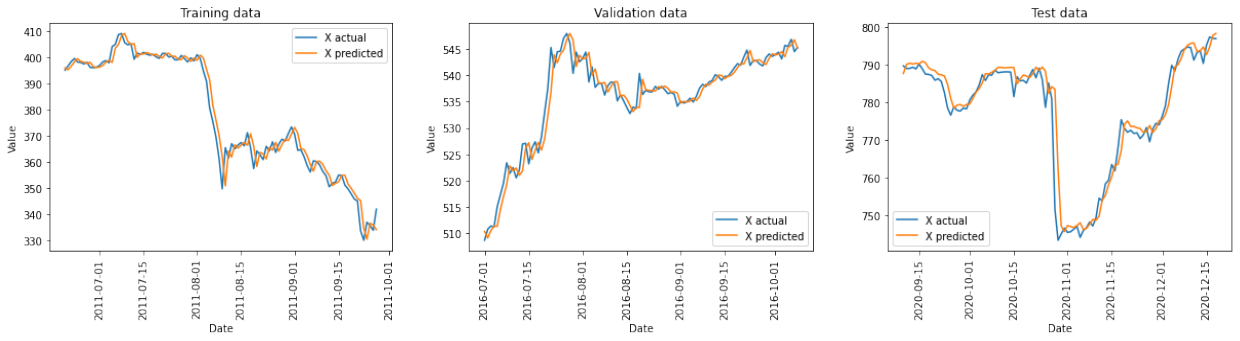


Figure 40. Last 100 actual and predicted values of training, validation and test data sets of the OMX Vilnius GI using SVR for the forecasting.

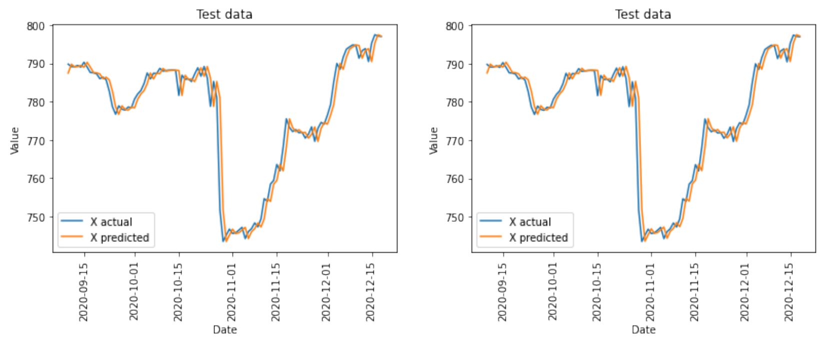


Figure 41. Last 100 actual and predicted values of test data set of the OMX Vilnius GI using ARIMA(1,0,0) (on the left) and ARIMA(0,1,0) (on the right).

	MSE	**RMSE**	**MAE**	**MASE**
MLP	10.8019	3.2866	1.8968	1.1465
RNN	9.9400	3.1528	1.8632	1.1262
SVR	11.1064	3.3326	1.9699	1.1907
ARIMA(1,0,0)	9.8331	3.1358	1.8716	1.2174
ARIMA(0,1,0)	9.8084	3.1318	1.8663	1.2140

Table 17. Final values of metrics used to evaluate the forecasting performance of models on the test data set of the OMX Vilnius GI.

It could be seen that based on the values of MSE and RMSE, ARIMA(0,1,0) shows the best forecasting performance, while SVR performs worst as might have been expected after looking at the graphical representation of predictions. However, based on the MAE and MASE metrics, RNN model outperforms all other models and, for instance, values of MASE show that all ML models perform better in a comparison to statistical models. By looking at the values of MASE, we could also see that all models perform worse than the naïve model in-sample and from the visual perspective it was noticed that with all models a small shift appears between the real and predicted values of the OMX Vilnius GI, which was also almost always seen in the examples of the simulation study. That confirms the fact that time series which are based on the random walk or has very similar behaviour to random walk (as analyzed market index), are very difficult to predict and all models are tend to obtain the most accurate prediction at a current time which is very close to the value of previous observation. Therefore, based on all results discussed in this part of the experiment, it can be concluded that ML models are able to slightly outperform traditional statistical models in the real world time series, which follow a random walk, forecasting, and, possibly, more difficult architecture or other ML models may be able to show even better forecasting performance, however the randomness in the behaviour of such time series could be the reason for the ML models to not make significantly better predictions than those, for example, which could be obtained using the simple naïve model.

4 Conclusions

The main idea of this work is to examine what differences are between nonstationary and nearly nonstationary time series forecasting performance using machine learning models. The literature review shows that the nearly nonstationary time series have similar properties and tendencies in the behaviour to the nonstationary ones and in practical applications of the machine learning models for the time series forecasting it is usual to divide time series only into two groups: stationary and nonstationary. However, the impact of changes in the degree of persistence in time series on forecasting using machine learning models is not considered and this is the area of interest.

One of the objectives in this study was to construct machine learning models (MLP, RNN and SVR) and test if they are able to achieve quite well performance on both nonstationary and nearly nonstationary time series forecasting. Results of the simulation study and the application of the real world data show that machine learning models could be a really effective tool to make quite accurate predictions for the nonstationary and nearly nonstationary time series, however, the accuracy of predictions obtained is very dependent on the parameters of models. Based on the simulation study where the constructed machine learning models have been applied for 100 AR(1) realizations forecasting it was noticed that models unequally respond (at least MLP and RNN) to the nonstationary and nearly nonstationary time series. It means that every example of a true random walk has to be considered as a special case due to the high randomness in the behaviour, while the time series which have only a similar behaviour to the random walk or are almost stationary do not have so many drastic changes and are more easily to predict. The analysis of the explosive time series also shows that this is a very unique case and also requires to be analyzed separately. Therefore, it could be assumed that the changes in the degree of persistence in time series could affect the forecasting performance of machine learning models.

Another objective of this work was to test if machine learning models could outperform traditional statistical models. Simulation results show that some of the machine learning models, as in this case RNN and SVR, are able to outperform commonly used statistical model ARIMA when the specific case of the nonstationary time series is predicted. However, the application of models for the more different realizations of nonstationary time series shows that the strong dependency on parameters of the machine learning models can cause high inefficiencies in forecasting what could mean that using machine learning models for the nonstationary time series forecasting for a long time it may be necessary to constantly update the model parameters what is really time consuming and is the weakness of the machine learning models. In the case of the nearly nonstationary time series forecasting, it was noticed that SVR (in the simulation study of a single AR(1) realization) and RNN (in the real world application) based on some forecasting performance metrics were able to slightly outperform traditional statistical models. However, overall, for the nearly nonstationary time series forecasting, it is more preferable to use traditional statistical models due to the fact that they show better or very similar forecasting performance to the machine learning models and are much easier to use. Possibly, more difficult machine learning models may be able to get more accurate predictions for the nearly nonstationary time series and outperform statistical models.

Taking all results into account, it could be concluded that the time series which are based on a random walk or follow a random walk are quite difficult to predict. For such time series, usually, the best prediction is very close to the value of the previous observation and it could be noticed that, even though all analyzed models in most cases are able to make quite accurate predictions, they always are tend to have smallest errors giving the prediction at the current time similar to the value of the observation a day before. Therefore, despite the fact that the changes in the degree of persistence in time series have an effect on the forecasting performance, for the time series with a high degree of persistence the simple naïve model could be quite good choice instead of difficult and time consuming constructions of more difficult, for example, machine learning, models. Also, it can be concluded that even though nowadays machine learning models usually are described as one of the most effective tools for the time series forecasting (especially for the nonstationary ones), they are tend to lose their power in the face of the randomness of the time series and usually are not able to make significantly better predictions in a comparison with traditional statistical models or even with the simplest naïve model.

References

[1] R. Achnata. "Long Term Electric Load Forecasting using Neural Networks and Support Vector Machines". In: *International Journal of Xomputer Science and Technology (IJCST)* 3.1 (2012). ISSN: 0976-8491 (Online).

[2] J. Adamowski and H. F. Chan. "A wavelet neural network conjunction model for groundwater level forecasting". In: *Journal of Hydrology* 407.1-4 (2011-09), pp. 28–40. DOI: `https://doi.org/10.1016/j.jhydrol.2011.06.013`.

[3] R. Adhiraki and R. K. Agrawal. "An Introductory Study on Time Series Modeling and Forecasting". In: *Nova York: CoRR* (2013).

[4] M. Awad and R. Khanna. "Support vector regression". In: *Efficient Learning Machines*. Springer, 2015, pp. 67–80. DOI: `10.1007/978-1-4302-5990-9_4`.

[5] M. Boden. "A guide to recurrent neural networks and backpropagation". In: *Dallas project* (2001).

[6] S. BuHamra, N. Smaoui and M. Gabr. "The Box-Jenkins analysis and neural networks: prediction and time series modeling". In: *Applied Mathematical Modelling* 27.10 (2003-10), pp. 805–815. DOI: `https://doi.org/10.1016/S0307-904X(03)00079-9`.

[7] M. Buscema. "Back Propagation Neural Networks". In: *Substance Use & Misuse* 33.2 (1998), pp. 233–270.

[8] J. Y. Campbell and P. Perron. "Pitfalls and Opportunities: What Macroeconomists Should Know about Unit Roots". In: *NBER macroeconomics annual* 6 (1991), pp. 141–201.

[9] L. Cao and Q. Gu. "Dynamic support vector machines for non-stationary time series forecasting". In: *Intelligent Data Analysis* 6.1 (2002), pp. 67–83. DOI: `10.3233/IDA-2002-6105`.

[10] L. J. Cao and F. E. H. Tay. "Support Vector Machine With Adaptive Parameters in Financial Time Series Forecasting". In: *IEEE Transactions on neural networks* 14.6 (2003), pp. 1506–1518.

[11] Q. Cao, B. T. Ewing and M. A. Thompson. "Forecasting wind speed with recurrent neural networks". In: *European Journal of Operational Research* 221.1 (2012-08), pp. 148–154. DOI: `https://doi.org/10.1016/j.ejor.2012.02.042`.

[12] C. Chatfield. *Time-series forecasting*. 71, 192 pp. CRC press, 2000.

[13] G. Chen. "A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation". In: *arXiv preprint arXiv:1610.02583* (2016). DOI: `arXiv:1610.02583v3`.

[14] S. M. Clarke, J. H. Griebsch and T. W. Simpson. "Analysis of support vector regression for approximation of complex engineering analyses". In: (2005).

[15] A. K. Dhamija and V. K. Bhalla. "Financial Time Series Forecasting: Comparison of Neural Networks and ARCH Models". In: *International Research Journal of Finance and Economics* 49 (2010), pp. 194–212. ISSN: 1450-2887.

[16] D. A. Dickey and W. A. Fuller. "Distribution of the Estimators for Autoregressive Time Series With a Unit Root". In: *Journal of the American Statistical Association* 74.366a (1979), pp. 427–431.

[17] F. X. Diebold and L. Kilian. "Unit-root tests are useful for selecting forecasting models". In: *Journal of Business & Economic Statistics* 18.3 (2000), pp. 265–273.

[18] B. Ding, H. Qian and J. Zhou. "Activation Functions and Their Characteristics in Deep Neural Networks". In: *2018 Chinese Control And Decision Conference (CCDC)*. IEEE. 2018, pp. 1836–1841.

[19] T. N. Do, V. H. Nguyen and F. Poulet. “Speed Up SVM Algorithm for Massive Classification Tasks”. In: *International Conference on Advanced Data Mining and Applications (ADMA)*. Springer, 2008, pp. 147–157. DOI: `https://doi.org/10.1007/978-3-540-88192-6_15`.

[20] K. Dzhaparidze, J. Kormos, T. Van der Meer and M. C. A. Van Zuijlen. “Parameter Estimation for Nearly Nonstationary AR(1) Processes”. In: *Mathematical and Computer Modelling* 19.2 (1994), pp. 29–41.

[21] G. B. A. Evans and N. E. Savin. “Testing For Unit Roots: 1”. In: *Econometrica: Journal of the Econometric Society* (1981), pp. 753–779.

[22] G. B. A. Evans and N. E. Savin. “Testing For Unit Roots: 2”. In: *Econometrica: Journal of the Econometric Society* (1984), pp. 1241–1269.

[23] J. Fan and Q. Yao. *Nonlinear Time Series: Nonparametric and Parametric Methods*. 18 p. New York, USA: Springer Science & Business Media, 2008.

[24] D. Fedorová and M. Arltová. “Selection of Unit Root Test on the Basis of Length of the Time Series and Value of AR(1) Parameter”. In: *Statistika* 96.3 (2016).

[25] M. D. Godfrey, C. W. J. Granger and O. Morgenstern. “The Random Walk Hypothesis of Stock Market Behavior”. In: *Kyklos* 17.1 (1964-02). 2, 6 pp., pp. 1–30. DOI: `10.1111/j.1467-6435.1964.tb02458.x`.

[26] R. Graf, S. Zhu and B. Sivakumar. “Forecasting river water temperature time series using a wavelet–neural network hybrid modelling approach”. In: *Journal of Hydrology* 578 (2019-11). DOI: `https://doi.org/10.1016/j.jhydrol.2019.124115`.

[27] C. Harpham and C. W. Dawson. “The effect of different basis functions on a radial basis function network for time series prediction: A comparative study”. In: *Neurocomputing* 69.16-18 (2006-10), pp. 2161–2170. DOI: `https://doi.org/10.1016/j.neucom.2005.07.010`.

[28] H. Hewamalage, C. Bergmeir and K. Bandara. “Recurrent Neural Networks for Time Series Forecasting: Current status and future directions”. In: *arXiv preprint arXiv:1909.00590* (2019). DOI: `https://doi.org/10.1016/j.ijforecast.2020.06.008`.

[29] S. L. Ho, M. Xie and T. N. Goh. “A comparative study of neural network and Box-Jenkins ARIMA modeling in time series prediction”. In: *Computers & Industrial Engineering* 42.2-4 (2002-04), pp. 371–375. DOI: `https://doi.org/10.1016/S0360-8352(02)00036-0`.

[30] N. Q. Hung, M. S. Babel, S. Weesakul and N. K. Tripathi. “An artificial neural network model for rainfall forecasting in Bangkok, Thailand”. In: *Hydrology & Earth System Sciences* 13.8 (2009), pp. 1413–1425. DOI: `https://doi.org/10.5194/hess-13-1413-2009`.

[31] R. J. Hyndman. “Another look at forecast-accuracy metrics for intermittent demand”. In: *Foresight: The International Journal of Applied Forecasting* 4.4 (2006), pp. 43–46.

[32] R. J. Hyndman and Y. Khandakar. “Automatic Time Series Forecasting: The forecast Package for R”. In: *Journal of Statistical Software* 26 (2008-07). DOI: `10.18637/jss.v027.i03`.

[33] I. Kaastra and M. Boyd. “Designing a neural network for forecasting financial and economic time series”. In: *Neurocomputing* 10 (1996), pp. 215–236.

[34] J. Kamruzzaman and R. A Sarker. “Forecasting of currency exchange rates using ANN: a case study”. In: *International Conference on Neural Networks and Signal Processing, 2003. Proceedings of the 2003*. Vol. 1. IEEE. 2003, pp. 793–797.

[35] F. Kaytez, M. C. Taplamacioglu, E. Cam and F. Hardalac. “Forecasting electricity consumption: A comparison of regression analysis, neural networks and least squares support vector machines”. In: *International Journal of Electrical Power & Energy Systems* 67 (2015), pp. 431–438. DOI: `https://doi.org/10.1016/j.ijepes.2014.12.036`.

[36] H. Kusdarwati and S. Handoyo. “System for Prediction of Non Stationary Time Series based on the Wavelet Radial Bases Function Neural Network Model”. In: *International Journal of Electrical and Computer Engineering* 8.4 (2018), pp. 2327–2337. ISSN: 2088-8708. DOI: `10.11591/ijece.v8i4.pp2327-2337`.

[37] H. Kusdarwati and S. Handoyo. “System for Prediction of Non Stationary Time Series based on the Wavelet Radial Bases Function Neural Network Model”. In: *International Journal of Electrical and Computer Engineering (IJECE)* 8.4 (2018-08), pp. 2327–2337. ISSN: 2088-8708. DOI: `10.11591/ijece.v8i4.pp2327-2337`.

[38] A. G. Libanio. “Unit roots in macroeconomic time series: theory, implications, and evidence”. In: *Nova Economia* 15.3 (2005), pp. 145–176.

[39] Z. C. Lipton, J. Berkowitz and C. Elkan. “A Critical Review of Recurrent Neural Networks for Sequence Learning”. In: *arXiv preprint arXiv:1506.00019* (2015). 12 pp. DOI: `arXiv:1506.00019v4`.

[40] C. Liu, S. C. H. Hoi, P. Zhao and J. Sun. “Online ARIMA Algorithms for Time Series Prediction”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[41] D. J. Livingstone, ed. *Artificial Neural Networks: Methods and Applications*. Springer, 2008. DOI: `10:1007/978-1-60327-101-1`.

[42] C. J. Lu, T. S. Lee and C. C. Chiu. “Financial time series forecasting using independent component analysis and support vector regression”. In: *Decision Support Systems* 47.2 (2009), pp. 115–125. DOI: `https://doi.org/10.1016/j.dss.2009.02.001`.

[43] J. C. Lu, D. X. Niu and Z. Y. Jia. “A study of short-term load forecasting based on ARIMA-ANN”. In: *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826)*. Vol. 5. IEEE. 2004, pp. 3183–3187.

[44] S. Makridakis and M. Hibon. “ARMA Models and the Box-Jenkins Methodology”. In: *Journal of Forecasting* 16.3 (1997), pp. 147–163.

[45] J. Markevičiūtė. “Asymptotic results on nearly nonstationary processes”. PhD thesis. Vilnius University, 2013.

[46] A. Mellit, A. M. Pavan and M. Benghanem. “Least squares support vector machine for short-term prediction of meteorological time series”. In: *Theoretical and Applied Climatology* 111.1-2 (2013), pp. 297–307. DOI: `https://doi.org/10.1007/s00704-012-0661-7`.

[47] H. Nie, G. Liu, X. Liu and Y. Wang. “Hybrid of ARIMA and SVMs for Short-Term Load Forecasting”. In: *Energy Procedia* 16 (2012), pp. 1455–1460. DOI: `10.1016/j.egypro.2012.01.229`.

[48] S. Ozturk and F. Ozturk. “Forecasting Energy Consumption of Turkey by Arima Model”. In: *Journal of Asian Scientific Research* 8.2 (2018), pp. 52–60. DOI: `10.18488/journal.2.2018.82.52.60`.

[49] W. Panichkitkosolkul and S.A. Niwitpong. “Multistep-ahead predictors for Gaussian AR(1) process with additive outlier following unit root tests”. In: *International Journal of Innovative Management, Information & Production* 1.1 (2010-12), pp. 49–55. ISSN: 2185-5439.

[50] G. Petnehází. “Recurrent Neural Networks for Time Series Forecasting”. In: *arXiv preprint arXiv:1901.00069* (2019-01).

[51] P. C. B. Phillips. “Towards a Unified Asymptotic Theory for Autoregression”. In: *Biometrika* 74.3 (1987), pp. 535–547.

[52] A. D. Pietersma. “Feature space learning in support vector machines through dual objective optimization”. PhD thesis. Faculty of Science and Engineering, 2010.

[53] D. K. Ranaweera, N. F. Hubele and A. D. Papalexopoulos. “Application of radial basis function neural network model for short-term load forecasting”. In: *IEE Proceedings-Generation, Transmission and Distribution*. Vol. 142. 1. IET, 1995-01.

[54] S. K. Safi. “Artificial Neural Networks Approach to Time Series Forecasting for Electricity Consumption in Gaza Strip”. In: *IUG Journal of Natural and Engineering Studies* 21.2 (2013), pp. 1–22. ISSN: 1726-6807. DOI: `http://www.iugaza.edu.ps/ar/periodical/`.

[55] I. Sanchez. “Efficient Forecasting in Nearly Non-stationary Processes”. In: *Journal of Forecasting* 21.1 (2002), pp. 1–26. DOI: `10.1002/for.812`.

[56] I. Sánchez and D. Peña. “Properties of predictors in overdifferenced nearly nonstationary autoregression”. In: *Journal of Time Series Analysis* 22.1 (2001), pp. 45–66.

[57] N. I. Sapankevych and R. Sankar. “Time Series Prediction Using Support Vector Machines: A Survey”. In: *IEEE Computational Intelligence Magazine* 4.2 (2009), pp. 24–38. DOI: `10.1109/MCI.2009.932254`.

[58] A. Shabri and Suhartono. “Streamflow forecasting using least-squares support vector machines”. In: *Hydrological Sciences Journal* 57.7 (2012), pp. 1275–1293. DOI: `https://doi.org/10.1080/02626667.2012.714468`.

[59] M. V. Shcherbakov, A. Brebels, N. L. Shcherbakova, A. P. Tyukov, T. A. Janovsky and V. A. Kamaev. “A Survey of Forecast Error Measures”. In: *World Applied Sciences Journal* 24.24 (2013), pp. 171–176. ISSN: 1818-4952. DOI: `10.5829/idosi.wasj.2013.24.itmies.80032`.

[60] G. Sheelapriya and R. Murugesan. “Stock price trend prediction using Bayesian regularised radial basis function network model”. In: *Spanish Journal of Finance and Accounting/Revista Española de Financiación y Contabilidad* 46.2 (2017), pp. 189–211. DOI: `http://dx.doi.org/10.1080/02102412.2016.1260859`.

[61] R. Sitte and J. Sitte. “Neural Networks Approach to the Random Walk Dilemmaof Financial Time Series”. In: *Applied Intelligence* 16.3 (2002), pp. 163–171.

[62] A. J. Smola and B. Schölkopf. “A tutorial on support vector regression”. In: *Statistics and Computing* 14.3 (2004), pp. 199–222.

[63] J. J. Sparks and Y. V. Yurova. “Comparative Performance of ARIMA and ARCH/GARCH Models on Time Series of Daily Equity Prices for Large Companies”. In: *2006 SWDSI proceedings* (2006), pp. 563–573.

[64] J. H. Stock. “VAR, error correction and pretest forecasts at long horizons”. In: *Oxford Bulletin of Economics and Statistics* 58.4 (1996), pp. 685–701.

[65] F. E. H. Tay and L. Cao. “Application of support vector machines in financial time series forecasting”. In: *Omega* 29.4 (2001-08), pp. 309–317. DOI: `https://doi.org/10.1016/S0305-0483(01)00026-3`.

[66] H. Wang and D. Hu. “Comparison of SVM and LS-SVM for Regression”. In: *2005 International Conference on Neural Networks and Brain*. Vol. 1. IEEE. 2005, pp. 279–283. DOI: `10.1109/ICNNB.2005.1614615`.

[67] J. Z. Wang, J. J. Wang, Z. G. Zhang and S. P. Guo. “Forecasting stock indices with back propagation neural network”. In: *Expert Systems with Applications* 38.11 (2011), pp. 14346–14355.

[68] Z. Wang, Y. Zhang and H. Fu. “Autoregressive Prediction with Rolling Mechanism for Time Series Forecasting with Small Sample Size”. In: *Mathematical Problems in Engineering* 2014 (2014).

[69] H. Yan and H. Ouyang. “Finansial Time Series Prediction Based on Deep Learning”. In: *Wireless Personal Communications* 102.2 (2018), pp. 683–700. DOI: https://doi.org/10.1007/s11277-017-5086-2.

[70] H. Yu, T. Xie, S. Paszczynski and B. M. Wilamowski. “Advantages of Radial Basis Function Networks for Dynamic System Design”. In: *IEEE Transactions on Industrial Electronics* 58.12 (2011-12), pp. 5438–5450.

[71] P. S. Yu, S. T. Chen and I. F. Chang. “Support vector regression for real-time flood stage forecasting”. In: *Journal of Hydrology* 328.3-4 (2006), pp. 704–716. DOI: https://doi.org/10.1016/j.jhydrol.2006.01.021.

[72] G. Zhang, B. E. Patuwo and M. Y. Hu. “Forecasting with artificial neural networks:: The state of the art”. In: *International Journal of Forecasting* 14.1 (1998-03), pp. 35–62. ISSN: 0169-2070. DOI: https://doi.org/10.1016/S0169-2070(97)00044-7.

[73] G. P. Zhang. “Time series forecasting using a hybrid ARIMA and neural network model”. In: *Neurocomputing* 50 (2003-01), pp. 159–175. DOI: https://doi.org/10.1016/S0925-2312(01)00702-0.

[74] Y. Zhang, Z. Chen, K. Chen and B. Cai. “Zhang Neural Network without Using Time-Derivative Information for Constant and Time-Varying Matrix Inversion”. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pp. 142–146.

[75] T. Zhou, F. Wang and Z. Yang. “Comparative Analysis of ANN and SVM Models Combined with Wavelet Preprocess for Groundwater Depth Prediction”. In: *Water* 9.10 (2017), p. 781. DOI: 10.3390/w9100781.

[76] B. Zhu, X. Shi, J. Chevallier, P. Wang and Y. Wei. “An Adaptive Multiscale Ensemble Learning Paradigm for Nonstationary and Nonlinear Energy Price Time Series Forecasting”. In: *Journal of Forecasting* 35.7 (2016), pp. 633–651. DOI: 10.1002/for.2395.