UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

“Machine Learning for Solar Energy Prediction”

TRABAJO FINAL DE GRADO

Autor/a:
Claudia Ferrer Martínez

Tutor/a:
José Chilo
Vicenç Almenar Terré

GANDIA, 2018

UNIVERSITY OF GÄVLE

FACULTY OF ENGINEERING AND SUSTAINABLE DEVELOPMENT
Department of Electronics, Mathematics and Natural Sciences

Machine Learning for Solar Energy Prediction

Claudia Ferrer Martínez

May 2018

Student thesis, Basic level (Bachelor degree), 15 hp
Bachelor's Degree in Electronics

Supervisor: José Chilo
Examiner: Edvard Nordlander

Preface

I would like to thank my supervisors José Chilo, professor at the University of Gävle (HiG), and Vicenç Almenar, professor at the Polytechnical University of Valencia (UPV), for giving me the opportunity to carry out this bachelor thesis. I would also like to thank my fellow students for all the help, support and shared moments during our university years. Thanks to the friends I met during my exchange semester in Gävle, for their support during the time spent working on this project and for making of the last semester of my bachelor a wonderful experience. Finally, I would like to thank specially my family and friends for their constant support during all the time.

Abstract

This thesis consists of the study of different Machine Learning models used to predict solar power data in photovoltaic plants.

The process of implement a model of Machine Learning will be reviewed step by step: to collect the data, to pre-process the data in order to make it able to use as input for the model, to divide the data into training data and testing data, to train the Machine Learning algorithm with the training data, to evaluate the algorithm with the testing data, and to make the necessary changes to achieve the best results.

The thesis will start with a brief introduction to solar energy in one part, and an introduction to Machine Learning in another part. The theory of different models and algorithms of supervised learning will be reviewed, such as Decision Trees, Naïve Bayer Classification, Support Vector Machines (SVM), K-Nearest Neighbor (KNN), Linear Regression, Logistic Regression, Artificial Neural Network (ANN).

Then, the methods Linear Regression, SVM Regression and Artificial Neural Network will be implemented using MATLAB in order to predict solar energy from historical data of photovoltaic plants. The data used to train and test the models is extracted from the National Renewable Energy Laboratory (NREL), that provides a dataset called "Solar Power Data for Integration Studies" intended for use by Project developers and university researchers. The dataset consist of 1 year of hourly power data for approximately 6000 simulated PV plants throughout the United States.

Finally, once the different models have been implemented, the results show that the technique which provide the best prediction is Linear Regression.

Key Words:

Machine Learning, Solar Energy, Forecasting, MATLAB, Supervised Learning, Regression Model

Resumen

El proyecto consistirá en el estudio de diferentes técnicas de Machine Learning aplicadas a la predicción de energía solar en plantas fotovoltaicas.

Se revisará el proceso de implementar un modelo de Machine Learning paso a paso: recoger el dataset, procesarlo para poder utilizarlo como entrada para el modelo, dvívidir los datos en datos para entrenar el modelo, y datos para evaluarlo, entrenar y evaluar el algoritmo, y hacer los cambios necesarios para conseguir los mejores resultados.

El proyecto comenzará con una breve introducción a la energía solar por una parte, y una introducción a Machine Learning por otra parte. Se hará una revisión de los diferentes modelos o algoritmos, en concreto de aprendizaje supervisado, tal como Árboles de Decisión, Clasificación Naïve Bayer, Máquinas de Vector de Soporte (SVM), K-Vecino más cercano (KNN), Regresión Lineal, Regresión Logística, Redes Neuronales Artificiales (ANN).

A continuación, los métodos Regresión Lineal, SVM y ANN serán implementados en MATLAB, con el fin de predecir energía solar partiendo de datos históricos de plantas fotovoltaicas. Los datos utilizados serán extraídos del Laboratorio Nacional de Energías Renovables (National Renewable Energy Laboratory (NREL)), de EEUU, que provee un dataset llamado "Solar Power Data for Integration Studies" destinado a desarrolladores de proyectos e investigadores universitarios que necesitan estimar la producción de energía para hipotéticas plantas solares. Este dataset consiste en datos de energía solar por hora durante un año, para aproximadamente 6000 plantas fotovoltaicas simuladas en EEUU.

Por último, una vez los diferentes modelos han sido implementados, los resultados muestran que la técnica que consigue mejores predicciones es Regresión Lineal.

Palabras Clave:

Machine Learning, Energía Solar, Predicción, MATLAB, Aprendizaje Supervisado, Modelo de Regresión

Table of contents

List of figures

List of tables

List of equations

1 Introduction

1.1 Aim of the project

Nowadays, renewable energies collected from renewable resources as sunlight or wind, are becoming more and more important, due to its minimum impact for the environment, as they produce a lower pollution and reduce CO2 emissions. However, renewable energies, such as photovoltaic energy, which is studied in this thesis, are not completely effective because the sun radiation varies constantly and electricity supply is needed in case there is no the expected solar energy to meet the demand for the electricity grid. That is a reason why solar energy is not commonly used in industries that need a continue power supply, due to the elevate costs of power bought at the last moment. Therefore, a good prediction system would be a great improvement in photovoltaic plants.

Machine Learning is a wide field of computer science that provides suitable techniques for making predictions. The main objective of this thesis is to study different techniques and algorithms of machine learning, and more specifically, supervised learning models, in order to see which one provides the best estimation of power produced by photovoltaic plants.

Thanks to new technologies such as Machine Learning, small improvements can be achieved in solar cells, which can lead into great improvements in a photovoltaic plant and make the photovoltaic energy systems more efficient.

1.2 Work plan

The work plan followed to complete this thesis is the following:

Total time: approximately 350 hours.

• 70 hours: search for information and research on the main fields of the project (machine learning, solar energy, software).

• 90 hours: theoretical analysis of machine learning techniques and writing of that section.

• 140 hours: Search for a suitable dataset, testing in MATLAB, implement the models and evaluate them.

• 50 hours: Writing the final project report.

2 Theory

2.1 Solar Energy

Nowadays, solar energy is the third most important renewable energy source, after hydro and wind power, and the second most deployed renewable technology in terms of global installed capacity [1]. The global capacity of photovoltaic plants installed around the world is expected to grow to over 900 GW by 2025, as shown in the graphic in the Figure 1 [2].

There is a continuous scientific development of technologies to improve the renewable energy systems in order to replace fossil fuels. Renewable energies produce minimum pollution and they have a better impact for the climatic change [3], which is one of the most important issues of this century. [4]

With regard to photovoltaic power evolution, the manufacturing of solar cells started with the aim of use them in space applications around 1950 [5], and nowadays, thanks to the researches and improvement of the photovoltaic technologies, as well as the reduction and more competitive building costs, solar cells are commonly used in industry and household.

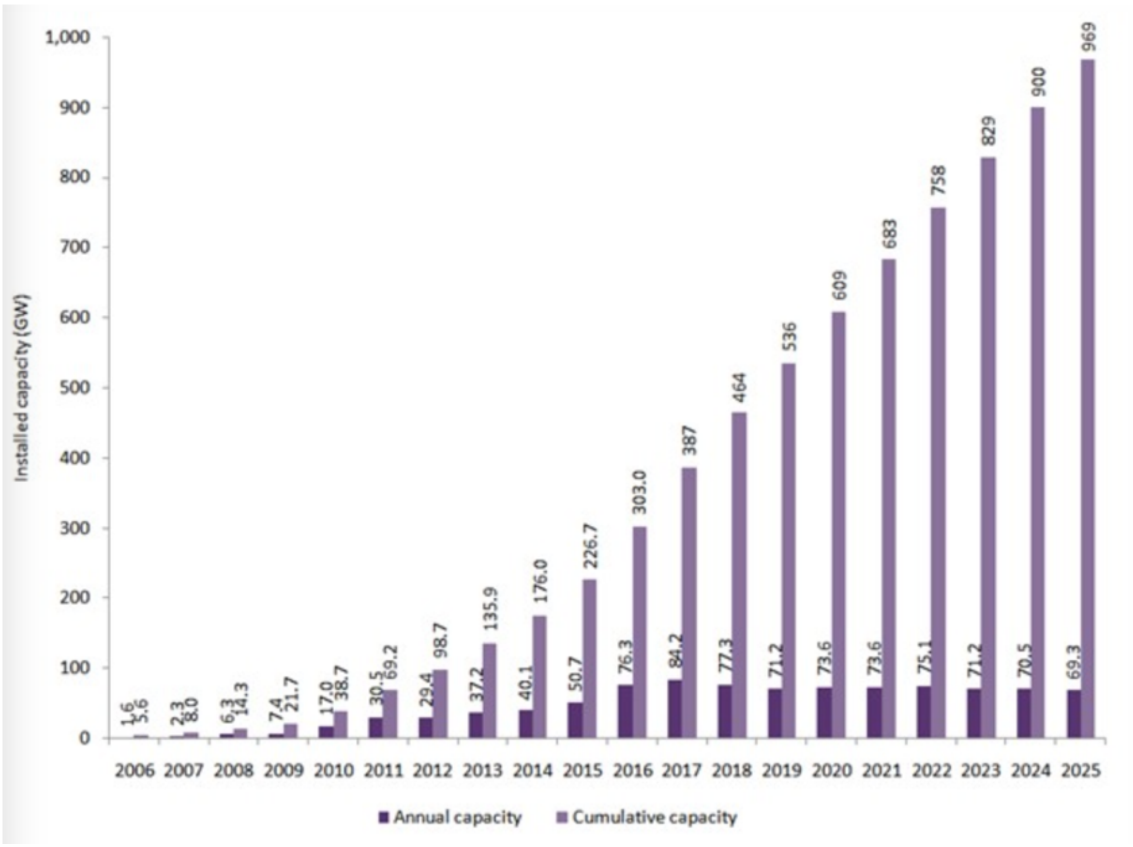


FIGURE 1: GLOBAL INSTALLED CAPACITY OF SOLAR POWER, FROM 2006 TO 2025 [2]

2.2 Machine Learning

2.2.1 Introduction to Machine Learning

Machine Learning is a field of Artificial Intelligence. It can be used in a wide variety of applications such as the following: classification of texts, speech recognition, computer vision tasks such as image recognition and face detection, self-driving vehicles, medical diagnosis.

Machine Learning is divided into two main techniques: unsupervised learning and supervised learning, shown in the Figure 2.

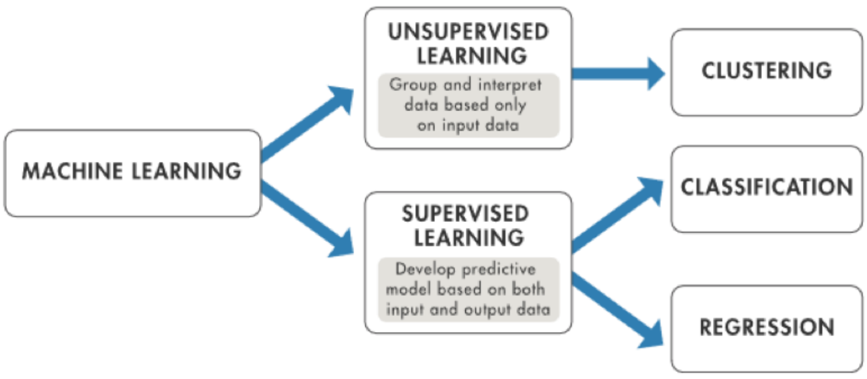

MACHINE LEARNING
UNSUPERVISED LEARNING
Group and interpret data based only on input data
SUPERVISED LEARNING
Develop predictive model based on both input and output data
CLUSTERING
CLASSIFICATION
REGRESSION

FIGURE 2: MACHINE LEARNING TECHNIQUES CLASSIFICATION [6]

On one hand, unsupervised learning uses only input data, not known output data, and try to find unknown patterns or structures in the input data to predict output data.

Unsupervised learning uses clustering, a technique that explores the data to find hiding natural patterns or groups. As it is shown in Figure 3, clustering consists on dividing the input data points into different groups. The data points contained in a group have similarities between them.

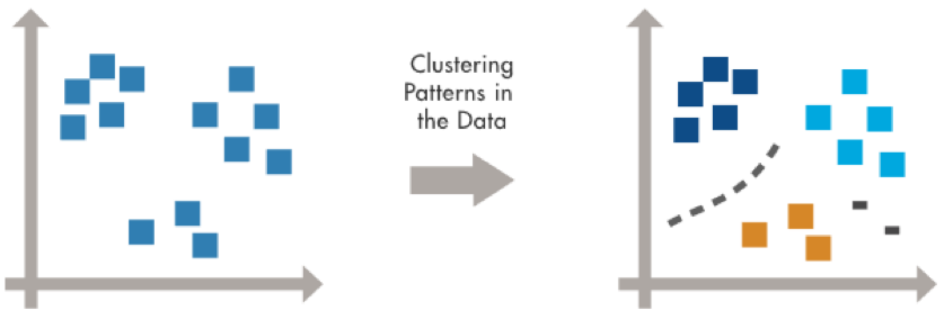


FIGURE 3: CLUSTERING FINDS NEW PATTERNS IN DATA [6]

Some applications of unsupervised learning algorithms are: object recognition or market analysis. Some algorithms used in clustering techniques are: K-Means, Gaussian Mixture, Neural Networks.

On the other hand, supervised learning consists on training a model where both input and output data are known and are used to predict new data. The process is to use an algorithm to learn the mapping function from the input to the output in order to use it with new input data to predict future output data.

Supervised learning uses two types of techniques to develop predictive models: classification and regression. These techniques will be explained in detail in the Section 2.2.2: Supervised Learning Algorithms.

There are some important steps in the process of building a machine learning model:

1. Collect the data: find a good dataset that allow us to use a prediction model and get the data we are looking for.

2. Preprocess the data: the data must have the correct format for the algorithm

3. Explore the data: see if there are insignificant values, errors, etc

4. Divide the data into training data and testing data

5. Train the algorithm with training data until a correct model with minimum errors is obtained.

6. Test the model with the testing data

7. Analyse results

In this thesis, the aim is to predict future data from historical data, i.e. is a type of time series prediction problem. Hence, it is a supervised learning problem. Supervised learning algorithms will be explained in the next section.

2.2.2 Supervised learning algorithms

2.2.2.1 Classification

The first technique used in supervised learning to develop predictive models is classification. Classification is used in cases when the data is discrete. The data is classified into concrete categories. Some applications of this technique are voice recognition, facial recognition, tumour patterns detection, etc.

The algorithms used to implement classification models are the following: Decision Trees, K-Nearest Neighbour, Logistic Regression, Naive Bayesian, Support Vector Machines, Artificial Neural Networks. [6] [7]

Decision Trees (DT)

Decision tree is one of the most used algorithms in classification problems. It is suitable for models whose output values are discrete, it is robust to noisy data, and capable of learning disjunctive expressions [7]. This mechanism consists on classifying instances by sorting them down the tree from the root to some leaf node, finally a tree having decision nodes and leaf nodes is obtained. The leaf node provides the classification of the instance. In Figure 4 it is shown an example of a Decision Tree, which classificates samples of weather conditions into two categories: "yes" if is posible to play tennis, and "no" if is not posible to play tennis.

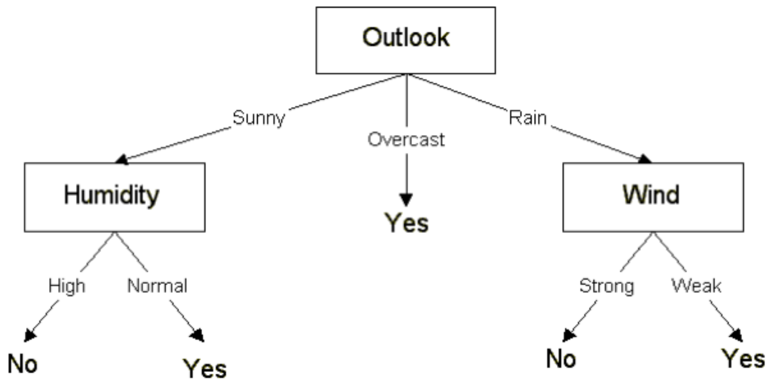


FIGURE 4: DECISION TREE OF PLAYING TENNIS [7]

In the case of classification, the algorithm used to built the decision tree uses entropy and information gain. In this context, entropy is used to calculate the homogeneity of a sample of data, it can be a value from 0 to 1, being 0 a completely homogeneous data and if the data can be equally divided, it has an entropy of 1. Information gain is the decreasement of entropy after the dataset is splitted. The steps of the algorithm are the following [8]:

1- Calculate entropy of the target.

2- Split the dataset based on different attributes and then calculate the entropy of each branch. Add it proportionally to obtain the total entropy after the split. Subtract total entropy after the split from the total entropy before the split to obtain information gain.

3- Attribute with largest information gain is the decision node.

4- A branch with entropy 0 is a leaf node.

5- A branch with entropy greater than 0 needs further splitting.

K-Nearest Neighbors (KNN)

K-NN is an algorithm used for many applications due to its simplicity and effectivity. It is a non-parametric technique, i.e., it does not make any assumptions on the data. A positive integer k is specified. For a new sample, the algorithm selects the k points on the dataset that have a similar pattern to the new sample (k-nearest neighbors). This task is done by calculating the distance between the new sample and all the existing samples on the dataset. For calculating the distance, it can be used the Euclidean (1), Manhattan (2) or Minkowski function (3) [8]:

$$\text{Euclidean function: } \sqrt{\sum_{i=1}^{k}(x_i - y_i)} \qquad (1)$$

EQUATION 1: EUCLIDEAN FUNCTION

$$\text{Manhattan function: } \sum_{i=1}^{k} |x_i - y_i| \qquad (2)$$

EQUATION 2: MANHATTAN FUNCTION

$$\text{Minkowski function: } \left(\sum_{i=1}^{k}(|x_i - y_i|^q)\right)^{1/q} \qquad (3)$$

EQUATION 3: MINKOWSKI FUNCTION

Once the k-nearest points are found, the most common class among these points will be the classification for the new sample.

Logistic Regression

This algorithm is used to solve binary classification problems, as the output can be 0 or 1.

The principle of this technique is the Logistic Function (4), where L is equal to the curve's maximum value, k is the steepness of the curve and x_0 is the x-value of the sigmoid's middle point. In Figure 5, an example of a logistic function is shown, where L=1, k=1 and x_0=0. [9]

This function is also called Sigmoid function due to its S-shaped curve. The logistic function take any real-valued number and transform them into values between the range of 0 and 1, but never exactly 0 or 1.

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}} \qquad (4)$$

EQUATION 4: LOGISTIC FUNCTION [9]

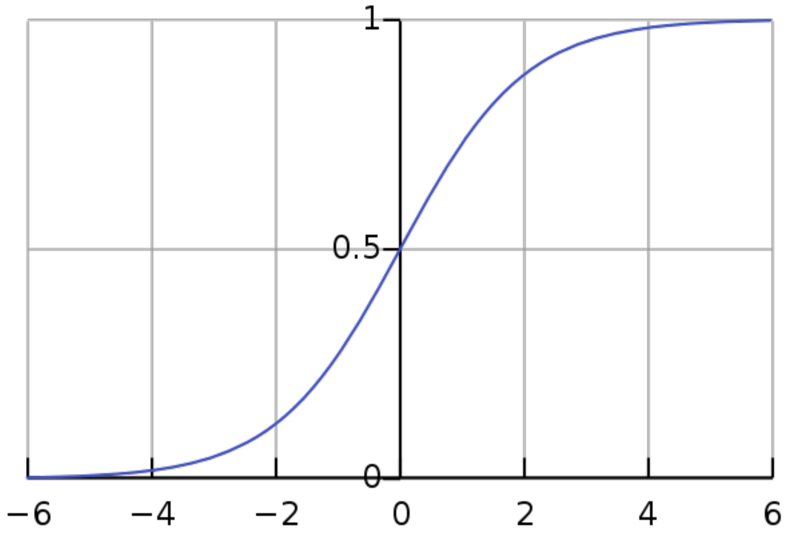


FIGURE 5: LOGISTIC FUNCTION [9]

The logistic regression classification method consists on calculating the probability of the input belonging to a concrete category. For example, obtaining an ouput of 0.7 in the logistic function would mean that there is 70% of probability of a datapoint belonging to a determined category. If the threshold is 0.5, the datapoint would be classified in that category.

Naïve Bayesian

The Naive Bayesian classifier is among the most effective algorithms known. It is based on the Bayes theorem, that consists on calculate the posterior probability P(H/E), from the prior probability P(h), together with P(E) and P(E/H), as it is shown in the Equation 5.

P(H) is the probability of hypothesis being true. It is known as the prior probability.

P(E) is the probability of the evidence, given no knowledge about the hypothesis.

P(E/H) is the probability of the evidence given that hypothesis is true.

P(H/E) is the probability of the hypothesis given that the evidence is there. It is known as the posterior probability.

$$P(H|E) = \frac{P(E|H)*P(H)}{P(E)} \qquad (5)$$

EQUATION 5: POSTERIOR PROBABILITY EQUATION [7]

Support Vector Machines

In this algorithm, the given training data is labeled in different classes, and the algorithm build an hyperplane or set of hyperplanes separating the classes with the maximum margin possible, which categorize new samples of data. There are some defining parameters in this technique [10]:

- Regularization parameter: for small values, the algorithm will look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points, see in Fig.6. For large values of this parameter, the algorithm will choose a smaller-margin hyperplane, see in Fig.7.

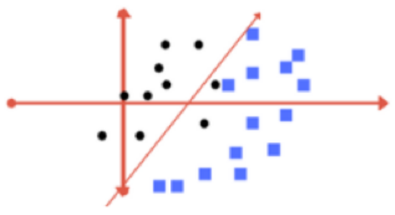


FIGURE 6: LOW REGULARIZATION PARAMETER

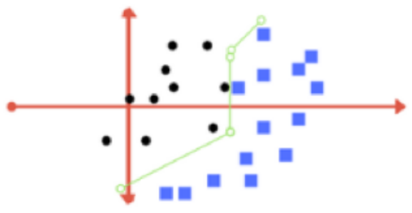


FIGURE 7: HIGH REGULARIZATION PARAMETER

- Gamma: High values of gamma, means only the closest points to plausible line are considered in calculation for the hyperplane, see in Fig.8. Low values of gamma, points far away from plausible separation line are considered in calculation for separation line, see in Fig.9.

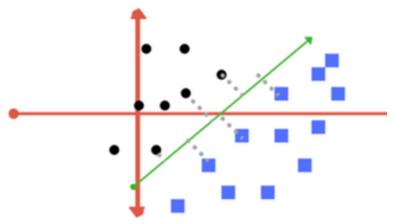


FIGURE 8: HIGH VALUES OF GAMMA

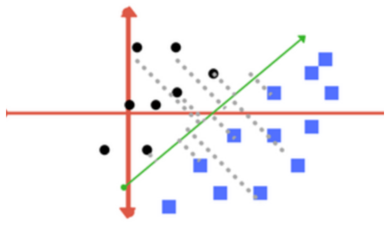


FIGURE 9: LOW VALUES OF GAMMA

- Kernel: The task of kernel is to take input data and transform it into the required form. To build the hyperplane, the algorithm can use different kernels, depending on the aim of the model, for example polynomial kernel for image processing or linear kernel for data classification.

- Margin: SVM has to find a good margin, i.e., when a separation line is equidistant and as far as possible from both classes.

Artificial Neural Networks

Neural networks are based on the performance of human neurons. Each neuron receives signals (input) and in function of these signals, it sends another signal or result to the network (output). In Fig.10 is shown the structure of an artificial neural network.

The neural network is distributed in layers, the first layer is the input, where it receives the signal, and the last layer is the output, it provides the results of the classification. The intermediate layers adjust the weights by comparing the output with the desired result, through back-propagation error.

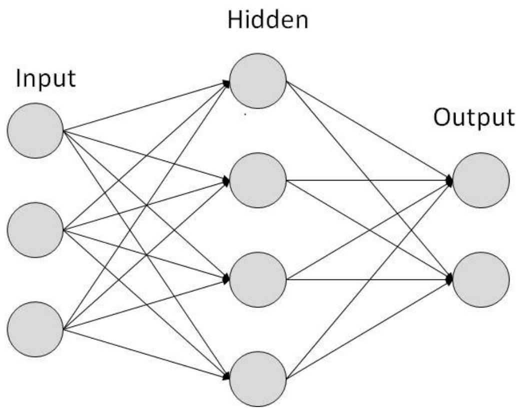


FIGURE 10: ARTIFICIAL NEURAL NETEWORK

2.2.2.2 Regression

The second technique used in supervised learning to develop predictive models is regression. In contrast with classification, regression algorithms work with real data, not discrete. Among some of their applications, they are used to predict continue re-sponses, as energy demand or temperature changes.

The idea of regression is to describe the output data as a lineal combination of the input data. Some algorithms used to implement regression models are: Linear Regression, Decision Trees, Support Vector Machines Regression and Artificial Neural Networks.

Linear Regression

Linear regression is one of the main algorithm used in supervised learning. The aim of linear regression is, using the least squares method, to find the coefficients of a linear combination that fit better to a group of disperse known points. The least-squares method minimizes the summed square of residuals.

Decision Trees

The mechanism is the same than Decision Trees use for classification, in Section 3.2.1. The difference is that when this method is used for regression techniques, the algorithm used to build the decision tree uses Standard Deviation Reduction (SDR). The steps are the following [7]:

1- First, the standard deviation of the target is calculated

2- Then, the datasets are divided on the different attributes, and the standard deviation of each branch is substracted from the standard deviation before the split. This is known as Standard Deviation Reduction (6):

$$SDR(T,X) = S(T) - S(T,X) \tag{6}$$

EQUATION 6: STANDARD DEVIATION REDUCTION

3- The attribute with the largest SDR will be the decision node.

4- The datasets are divided based on the values of selected attributes. If the standard deviation of a branch is more than 0, is divided again.

5- Keep the process until all the data is processed.

Support Vector Machines Regression

Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm. The difference between SVM and SVMR is that SVMR works by doing a regression from the classifier.

To do that, it performs a non-linear mapping of the training data to a higher dimension space over a kernel, where it is possible to do a linear regression. The selection of the kernel will define a more or less efficient model.

Artificial Neural Network

It can be also used for regression techniques, to predict continuous values.

There are several types of neural networks, the models studied in this thesis are: non linear autoregressive with external input and nonlinear autoregressive, explained in the Section 3.3.3.

3 Process and results

In this thesis the aim is to predict future solar energy data from historical data, hence the model will be implemented with regression techniques from supervised learning. These techniques are implemented in MATLAB, the full code of the different tests is found in the Appendix.

3.1 Dataset

The dataset used in this thesis is provided by the NREL (National Renewable Energy Laboratory). In the website of NREL, it is provided data from every state of USA [11]. For this thesis the dataset of Texas will be used, as this state has the largest solar potential in the country.

The dataset consists on hourly solar power data values during one year in 52 different locations in the state of Texas. As it is said in its official website, it is hypothetical data, intended for use by energy professionals, project developers and university researchers who perform solar integration studies and need to estimate power production from hypothetical solar plants [11].

However, in this thesis the dataset will be used as historical data in order to predict the future solar energy that will be produced by a determined photovoltaic system. The data from the different locations will be put together in the same dataset, as all the locations are from the same state and they have similar range of power values, data of different locations will be helpful for the machine learning models.

The total dataset consists on a matrix of 8760 rows and 54 columns, the first column is the month, the second one the day of the month, and the rest of the columns are the different locations. The rows are the hourly data, as it is one year, there are 365x24= 8760 points of solar energy data. The solar energy data is in MW.

In addition, as the maximum values in each location are different, the data has been normalized to values from 0 to 1, in order to improve the efficiency on the different models. The data before and after normalization is shown in the plots in Figures 11 and 12.

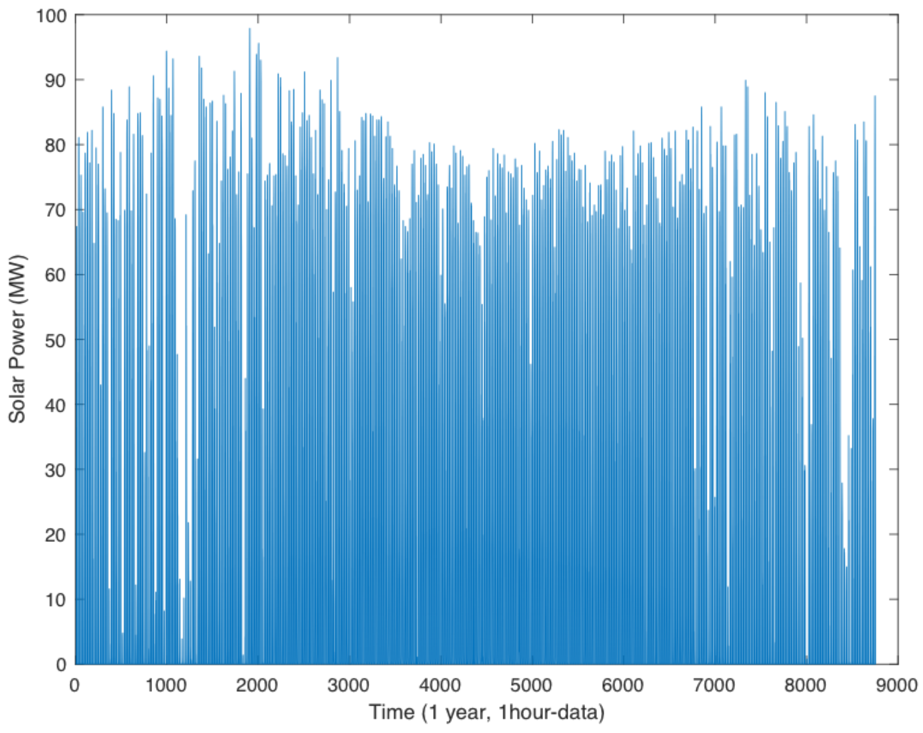


FIGURE 11: DATASET

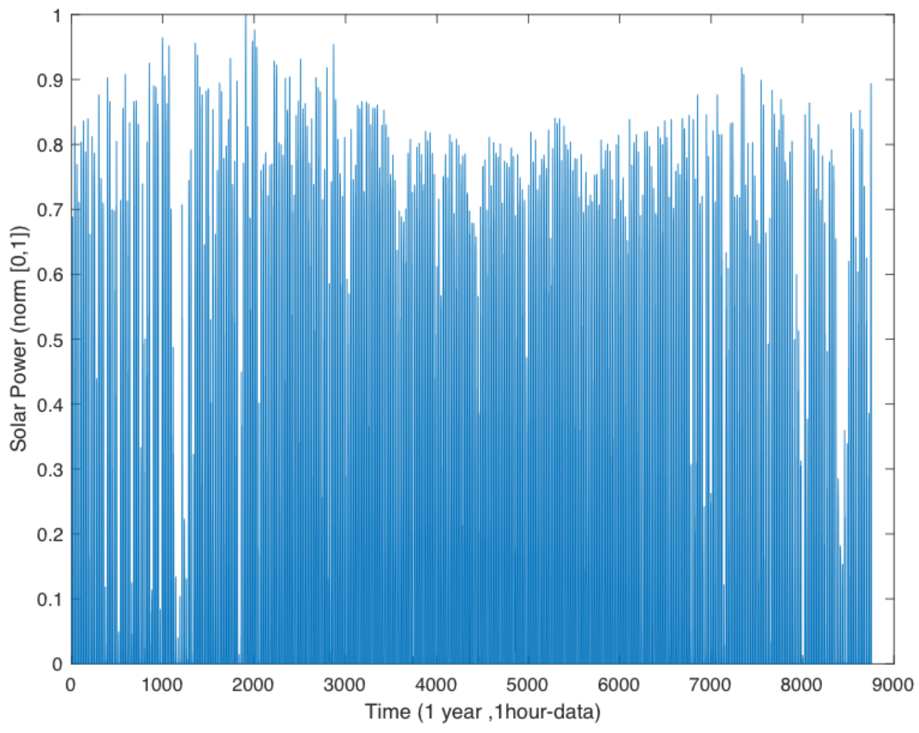


FIGURE 12: DATASET OF FIG.11 NORMALIZED FROM 0 TO 1

For the machine learning algorithms, it is needed two types of data: training data and testing data. In this case, the dataset has been divided into 75% training data and 25% testing data. The algorithms will be first trained with the training data, i.e., it is provided a series of predictors (input) and the known results (output), and the model will work with this data to find a relation between the predictors and the results. As this is a time-series problem, the predictors are past data values, and additionally, the month and day of the month of each data value.

Once the relation is obtained, if it is not enough accurate, the model can be retrained until the correct results are obtained. After that, the algorithms are tested with the testing data. In this step, it is only provided to the algorithm the input data, to test if the model can correctly make predictions. Once the predictions are done, as the real output data of the test data is on the data set, the predicted data and real data is compared in order to see which model is better.

The results of every model will be discussed comparing the Root Mean Square Error (RMSE) (7) of the model performance in the training phase, as well as the RMSE of the predicted data vs. real data in the testing phase.

$$RMSE = \sqrt{(predicted - observed)^2} \qquad (7)$$

EQUATION 7: RMSE

3.2 Software

In this thesis, the software MATLAB will be used to implement the different models of Machine Learning, as it has been used in many different subjects during the bachelor. MATLAB provides a series of tools and apps for Super- vised Learning Machine Learning methods:

- Statistics and Machine Learning Toolbox: It provides functions and apps to describe, analyse and model data. More specifically, it provides supervised and unsupervised machine learning algorithms, including support vector machines (SVMs), decision trees, k-nearest neighbour and k-means among others [12]. Regression and classification algorithms that allow you to build predictive models. Among the applications provided by this toolbox, the app Regression Learner will be used, to train regression models to predict data, as Linear Regression, Decision Trees and Support Vector Machines.

- Neural Network Toolbox: It provides algorithms, pretrained models, and apps to create, train, visualize, and simulate both shallow and deep neural networks. You can perform classification, regression, clustering, dimensionally reduction, time-series forecasting, and dynamic system modelling and control [13]. The app Neural Net Time Series will be used in this thesis. This app allows you to solve nonlinear time series problems by training dynamics neural networks, selecting the training data. After training the network, evaluate its performance using mean squared error and regression analysis.
  After training the models, the algorithm can be extracted in MATLAB script format, so it can be tested with new data.

3.3 Experiments

This section is divided into three main parts: Linear Regression, Support Vector Machines Regression and Artificial Neural Networks. These are the three algorithms or techniques that have been tested in MATLAB. In the end of this section, there is an additional fourth part performing some possible changes in the dataset in order to improve the results.

As the Figure 13 shows, the process to follow in building every Machine Learning model is the same. The first step is training the model with known data and known responses, and the second step is to test the model with new data to obtain predicted responses.

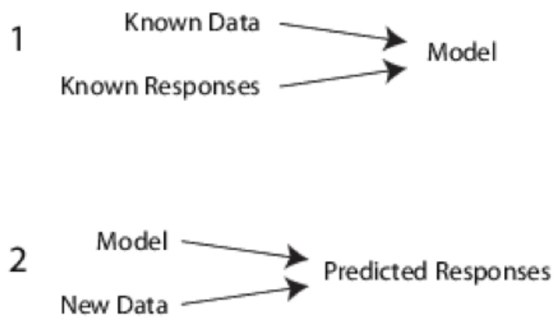


FIGURE 13: MAKING PREDICTIONS WITH A MACHINE LEARNING MODEL

3.3.1 Linear Regression (LR)

The first experiment performed in MATLAB is a linear regression model. The Statistics and Machine Learning Toolbox in MATLAB, provides an app called “Regression Learner”. This app is useful to train regression models in order to predict data using supervised machine learning.

Two different models will be implemented in this section. First, the model will be trained to predict future data in one location, using historical data from the nearest locations to that one, and secondly, data from every location will be used to predict data in one location.

First LR model: prediction using data from nearest locations

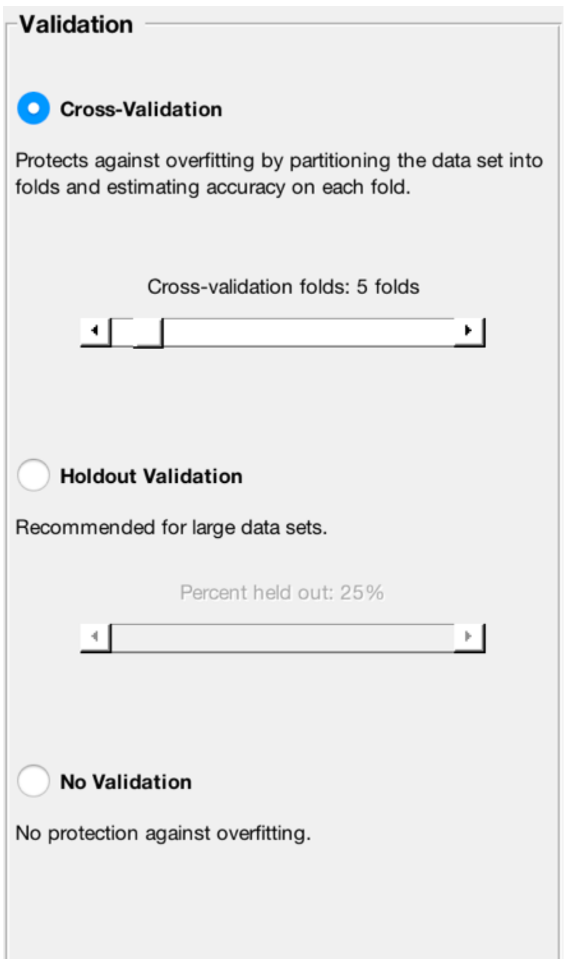
The model will be trained to predict data in one location using historical data from various locations around that one. The coordinates of each location used in this model are shown in the Table 1. The location whose data is predicted will be the one in column 52 of the dataset.

TABLE 1: PREDICTORS AND RESPONSE

	Predictor data				Response
Nº column of the dataset	50	51	53	54	52
Coordinates	(35.45, -101.85)	(35.75, -102.95)	(36.05, -102.95)	(36.25, -102.95)	(35.85, -102.85)

To start building the model, it is necessary to upload the training dataset, and select what variables are the predictors, and what variables are the desired result. It is necessary also to choose a validation scheme to examine the predictive accuracy of the models and protect against overfitting. Overfitting means when the model trains too much the data, that even the noise or undesired variations in data are learned as features by the model.

It is important to choose the same validation scheme in every model to correctly compare them in the end. There are three different options for validation: Cross-Validation, Holdout Validation and No Validation, see in Fig.14. The last option is not recommended as it does not protect against overfitting. Holdout Validation is appropriate for large data sets, thus, the validation scheme used in every test of this thesis, will be Cross-Validation.



Validation

[✓ Cross-Validation]

Protects against overfitting by partitioning the data set into folds and estimating accuracy on each fold.

Cross-validation folds: 5 folds

Holdout Validation

Recommended for large data sets.

Percent held out: 25%

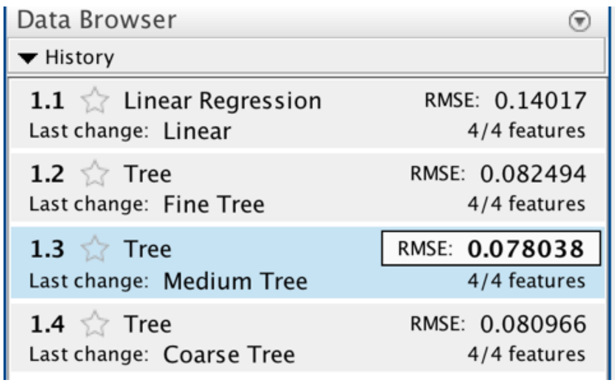
No Validation

No protection against overfitting.

FIGURE 14: VALIDATION OPTIONS IN MATLAB

In the case of Cross-Validation, it is necessary to determine the number of folds (k) in which the dataset is divided. For each fold, the model is trained using the out-of-fold observations, and the model assessed its performance using in-fold data. Finally the average test error is calculated over all folds. The default option will be used for this thesis, 5-fold cross-validation (k=5), which protects agains overfitting.

Next, when the data and the configuration of the model is determined, in the app it is selected the option that allows to try different architectures of the model at the same time: Linear Regression, Fine Tree, Medium Tree and Coarse Tree. Once the results of RMSE of every architecture are obtained, the model with the smallest error is selected, and it is generated the MATLAB code in order to test it with new data and make predictions.
In this case, the model with better performance is the Medium Tree with RMSE=0.078038, as the Figure 15 shows.


Data Browser

History

1.1	Linear Regression	RMSE: 0.14017
	Last change: Linear	4/4 features
1.2	Tree	RMSE: 0.082494
	Last change: Fine Tree	4/4 features
1.3	Tree	RMSE: 0.078038
	Last change: Medium Tree	4/4 features
1.4	Tree	RMSE: 0.080966
	Last change: Coarse Tree	4/4 features

FIGURE 15: RMSE OF DIFFERENT TRAINED CONFIGURATIONS (FIRST LR MODEL: PREDICTION USING DATA FROM NEAREST LOCATIONS)

To obtain more information about the performance of the model, In Figure 16 it is shown the Predicted vs. Actual Response plot. The diagonal black line shows how would be the perfect prediction, in which the predicted and true response are the same. The closest to the diagonal line are the observations (blue points), the better is the model. In this case, most of the points are located near the diagonal line, although there are some observations with error.

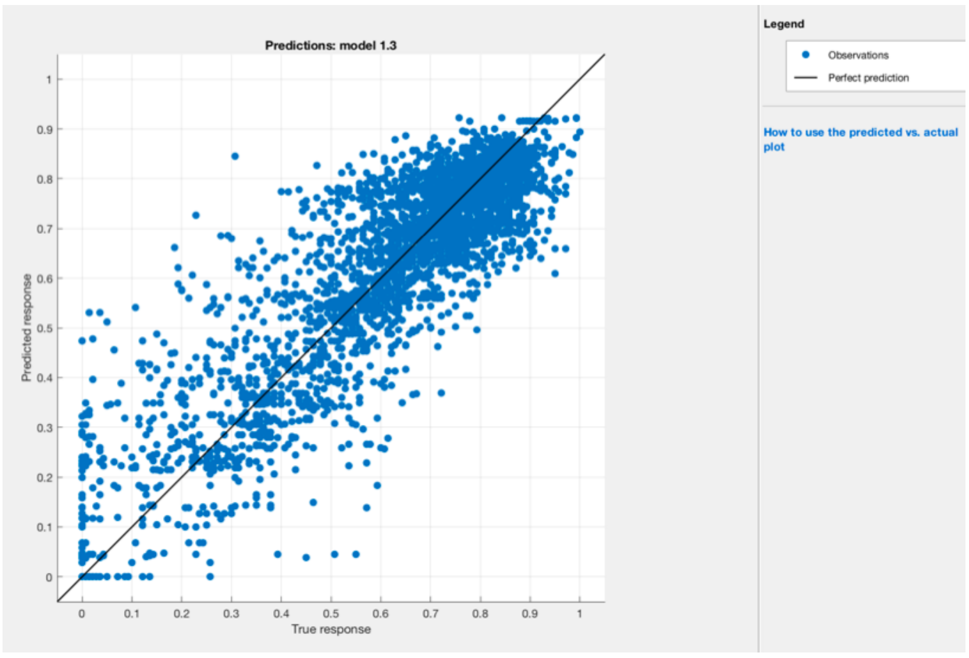


FIGURE 16: PREDICTED VS. ACTUAL RESPONSE (FIRST LR MODEL: PREDICTION USING DATA FROM NEAREST LOCATIONS)

The next step is to generate the MATLAB code for the Tree model in order to test it with new data and make predictions. The code of every model can be found in the appendix. After testing the model, a plot is generated to compare the predictions with the real data to evaluate its performance. The Figure 17 shows the graphic of 100 hours, to see more in detail the curve, and the Figure 18 shows predicted data for 1 month (720 hours). The error between the predicted data and real data is calculated: RMSE=0.0081.

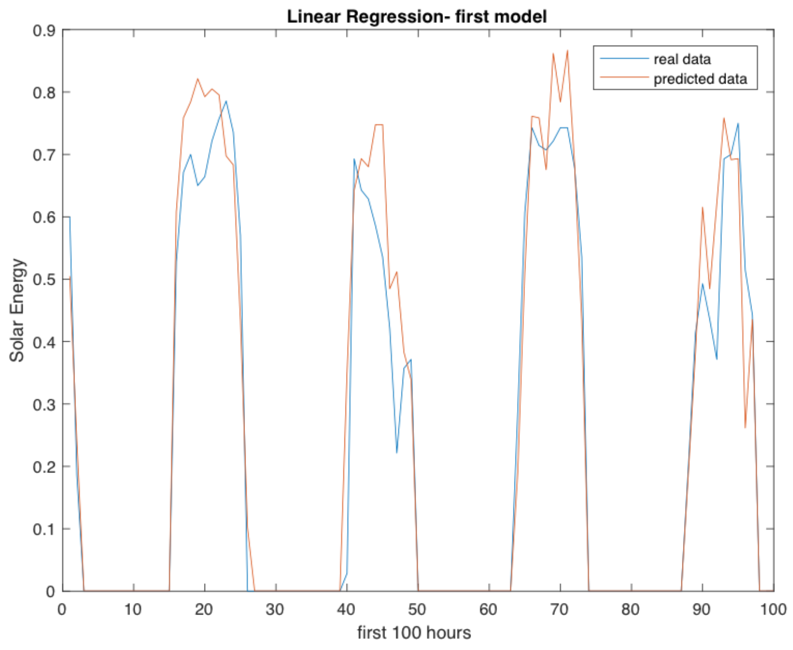


FIGURE 17: PREDICTED DATA VS. REAL DATA, EXTRACT OF 100 HOURS (FIRST LR MODEL: PREDICTION USING DATA FROM NEAREST LOCATIONS)

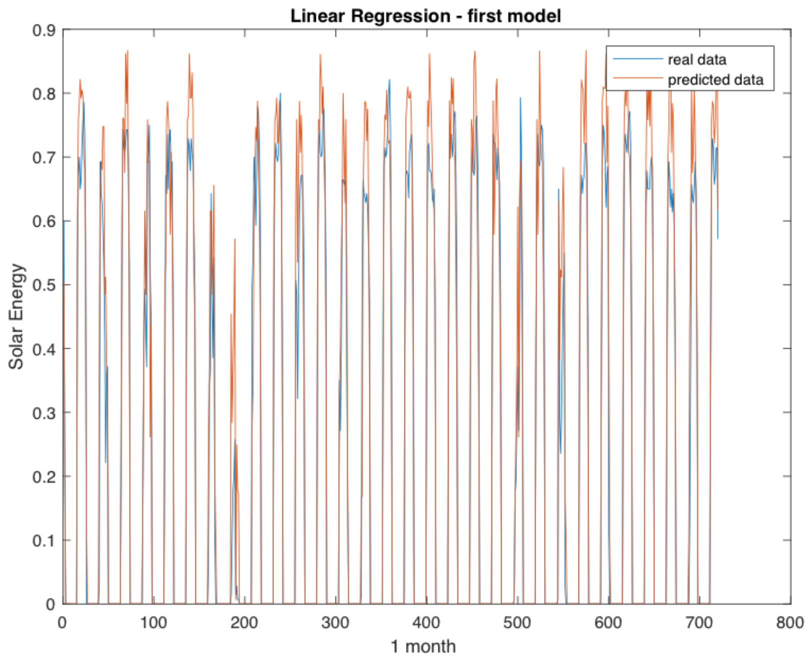
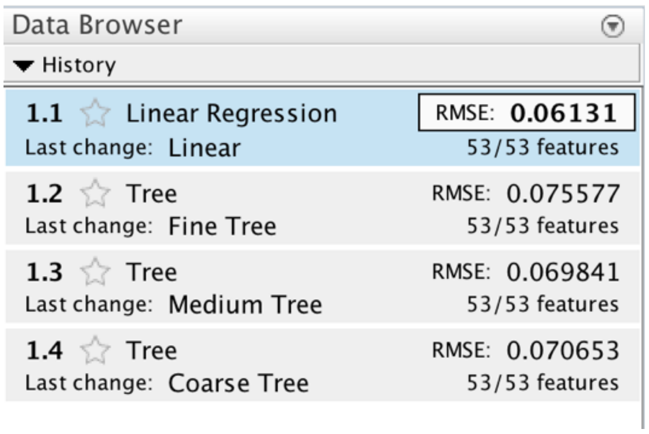


FIGURE 18: PREDICTED DATA VS. REAL DATA, EXTRACT OF 1 MONTH (FIRST LR MODEL: PREDICTION USING DATA FROM NEAREST LOCATIONS)

Second LR model: prediction using data from every location

In this case, the model will predict the data in one location, using historical data from every other location. The location whose data is predicted is the same than the previous one (column 52, coordinates: (35.85, -102.85)) in order to compare properly both models.

The architecture with better performance in this case is Linear Regression, with RMSE=0.06131, see in Fig. 19.



Data Browser

History

1.1 Linear Regression — RMSE: 0.06131
Last change: Linear — 53/53 features

1.2 Tree — RMSE: 0.075577
Last change: Fine Tree — 53/53 features

1.3 Tree — RMSE: 0.069841
Last change: Medium Tree — 53/53 features

1.4 Tree — RMSE: 0.070653
Last change: Coarse Tree — 53/53 features

FIGURE 19: RMSE OF DIFFERENT TRAINED CONFIGURATIONS (SECOND LR MODEL: PREDICTION USING DATA FROM EVERY LOCATION)

The Figure 20 shows the graphic of 100 hours, to see more in detail, and the Figure 21, 1 month (720 hours). The error between the predicted data and real data is calculated: RMSE=0.0016.

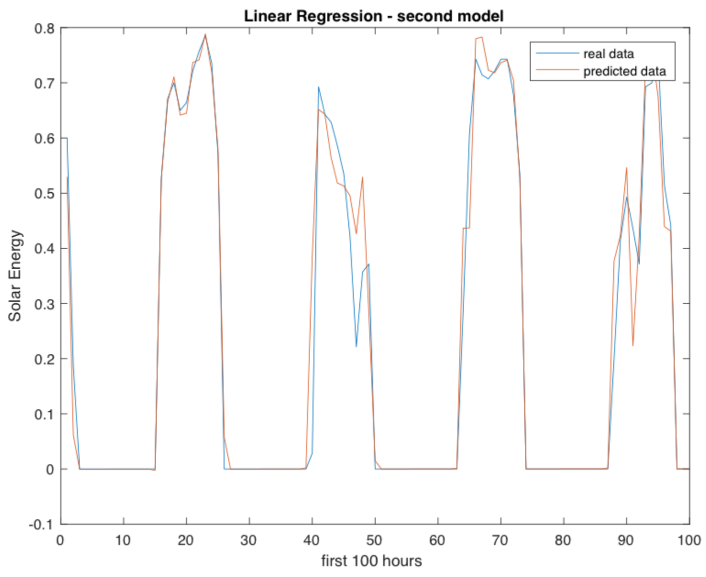


FIGURE 20: PREDICTED DATA VS. REAL DATA, EXTRACT OF 100 HOURS (SECOND LR MODEL: PREDICTION USING DATA FROM EVERY LOCATION)

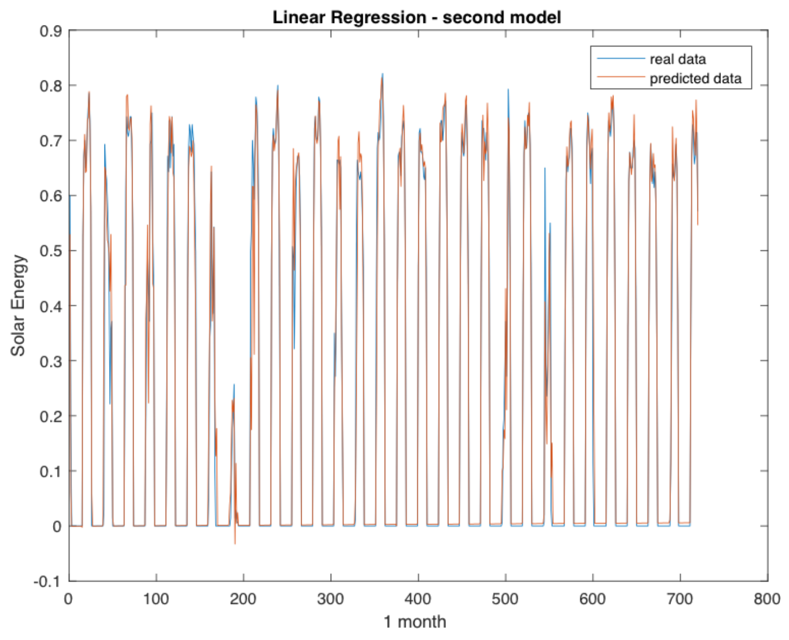


Figure 21: Predicted data vs. real data, extract of 1 month (Second LR model: prediction using data from every location)

The Table 2 shows the comparison of both tested models of Linear Regression, with the results expressed in RMSE of the model in the training step, and RMSE of predicted/real data in the testing step.

Table 2: Comparison of results of Linear Regression models

	Prediction using data from nearest locations	Prediction using data from every location
RMSE model	0.0791	0.0610
RMSE predicted/real data	0.0081	0.0016

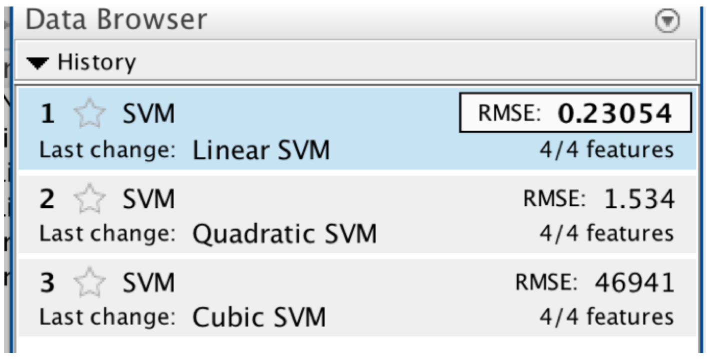
3.3.2 Support Vector Machines Regression (SVMR)

The second experiment in MATLAB consists on trying the technique Support Vector Machines Regression in the app Regression Learner.

First SVMR model: predictions using data from nearest locations

The process for this section is the same, to select the training data, the predictors and the desired response and build the model. In this case, it has been compared three different SVM models, the first one, using a linear kernel, the second one, using a quadratic kernel, and the third one, using a cubic kernel.

As shown in the Figure 22, the SVM model with linear kernel has a better performance than the quadratic and cubic kernel. The model is exported to be tested with the testing data. The graphics comparing predicted vs. real data are on the appendix.



Data Browser

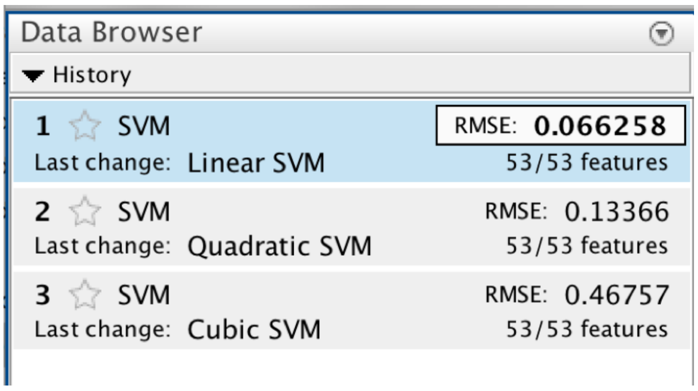
History

1 SVM	RMSE: 0.23054
Last change: Linear SVM	4/4 features
2 SVM	RMSE: 1.534
Last change: Quadratic SVM	4/4 features
3 SVM	RMSE: 46941
Last change: Cubic SVM	4/4 features

FIGURE 22: RMSE OF TRAINED MODEL WITH DIFFERENT KERNELS (FIRST SVMR MODEL: PREDICTIONS USING DATA FROM NEAREST LOCATIONS)

Second SVMR model: predictions using data from every location

As in the previous model, the model with the smaller error is Linear SVM, see in Fig.23.



Data Browser

History

1 SVM	RMSE: 0.066258
Last change: Linear SVM	53/53 features
2 SVM	RMSE: 0.13366
Last change: Quadratic SVM	53/53 features
3 SVM	RMSE: 0.46757
Last change: Cubic SVM	53/53 features

FIGURE 23: RMSE OF TRAINED MODEL WITH DIFFERENT KERNELS (SECOND SVMR MODEL: PREDICTIONS USING DATA FROM EVERY LOCATION)

The results of both SVMR models, expressed in RMSE, are shown in Table 3.

TABLE 3 : COMPARISON OF RESULTS OF BOTH SVMR MODEL

	SVM linear kernel, with data from nearest locations	SVM linear kernel, with data from every location.
RMSE Model	0.2370	0.0664
RMSE Predicted/real data	0.0792	0.0026

3.3.3 Artificial Neural Network (ANN)

In this section, it is tested the Machine Learning technique Artificial Neural Networks. The app Neural Network Time Series in the toolbox Neural Network is used. This app allows you to solve time series problems, which is the case, as the predictors are historical data.

There are three different type of ANN problems. The first one, Nonlinear Autoregressive with External Input (NARX) predict series y(t) given d past values of y(t) and another series x(t), see in Fig.24. The second one, Nonlinear Autoregressive (NAR) predict series y(t) given d past values of y(t), see in Fig.25, and the third one, Nonlinear Input-Output predict series y(t) given past values of series x(t), see in Fig.26, but is not performed in this thesis due to is not as accurate as the other ones, and if the past data is known, is not necessary to use this model.

Thus, this section is divided in two parts: the first part, using the NARX model, and the second done, using the NAR model.

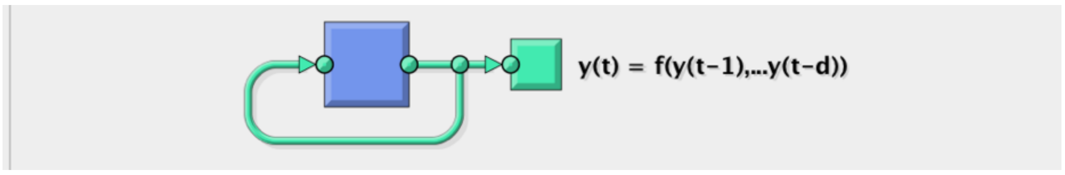


FIGURE 24: NONLINEAR AUTOREGRESSIVE WITH EXTERNAL INPUT (NARX)

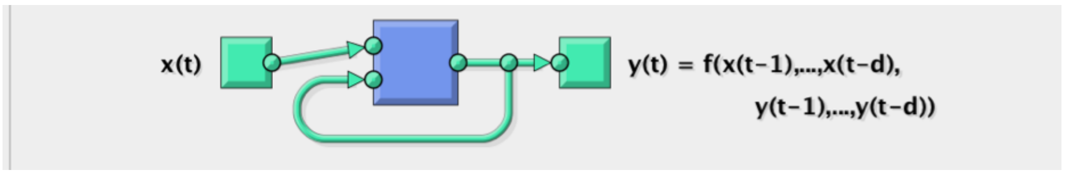


FIGURE 25: NONLINEAR AUTOREGRESSIVE (NAR)

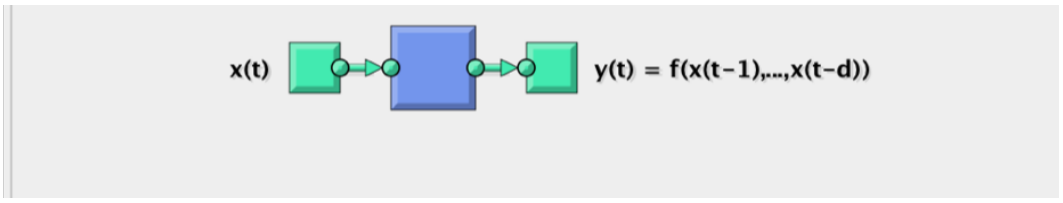
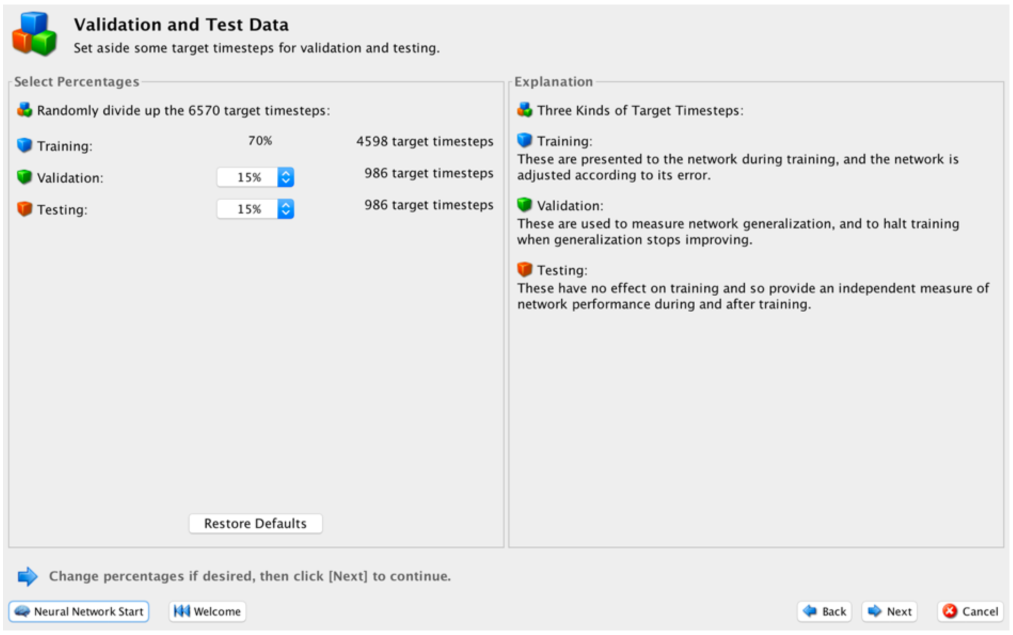


FIGURE 26: NONLINEAR INPUT OUTPUT

First ANN model: prediction using data from every location

The first model of Artificial Neural Network is the model NARX (Nonlinear Autoregressive with External Input). The model will predict data in one location, using historical data from every location.

In this model, the procedure is to provide the training dataset and the app Neural Network Time series divides the dataset into training (70 %), validation (15 %) and testing (15 %) in order to build and train the model, this process is shown in Figure 27.


Validation and Test Data
Set aside some target timesteps for validation and testing.

Select Percentages

Randomly divide up the 6570 target timesteps:

Training:	70%	4598 target timesteps
Validation:	15%	986 target timesteps
Testing:	15%	986 target timesteps

Explanation

Three Kinds of Target Timesteps:

Training:
These are presented to the network during training, and the network is adjusted according to its error.

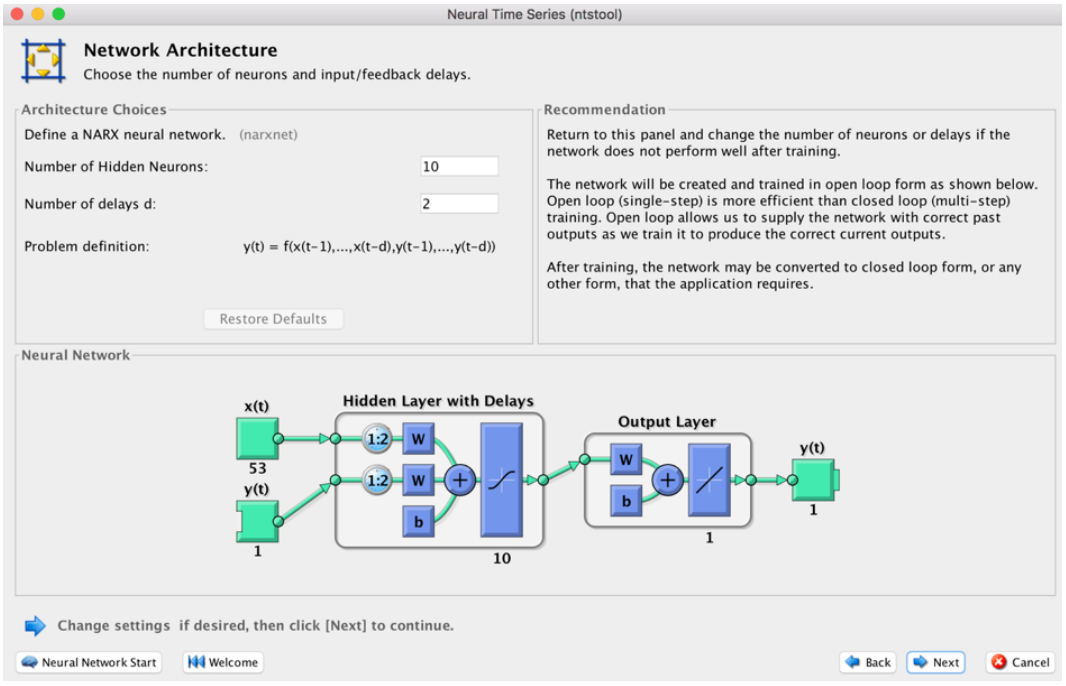
Validation:
These are used to measure network generalization, and to halt training when generalization stops improving.

Testing:
These have no effect on training and so provide an independent measure of network performance during and after training.

Change percentages if desired, then click [Next] to continue.

FIGURE 27: DIVIDING THE DATASET INTO TRAINING, VALIDATION AND TESTING

The next step is to determine the number of hidden neurons of the network, and the number of delays. For this first example, the neural network is configured with 10 hidden neurons and a delay of 2 time-steps, see in Figure 28.


Neural Time Series (ntstool)

Network Architecture
Choose the number of neurons and input/feedback delays.

Architecture Choices

Define a NARX neural network. (narxnet)

Number of Hidden Neurons: 10

Number of delays d: 2

Problem definition: y(t) = f(x(t-1),...,x(t-d),y(t-1),...,y(t-d))

Recommendation

Return to this panel and change the number of neurons or delays if the network does not perform well after training.

The network will be created and trained in open loop form as shown below. Open loop (single-step) is more efficient than closed loop (multi-step) training. Open loop allows us to supply the network with correct past outputs as we train it to produce the correct current outputs.

After training, the network may be converted to closed loop form, or any other form, that the application requires.

Change settings if desired, then click [Next] to continue.

FIGURE 28: CHOOSING NUMBER OF HIDDEN NEURONS AND NUMBER OF DELAYS

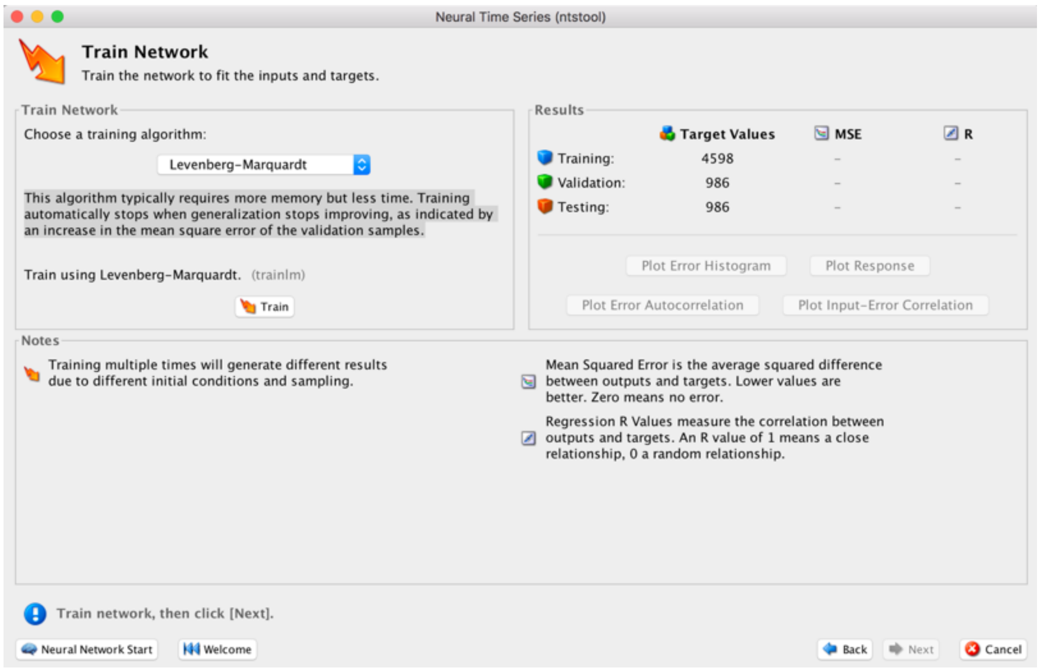
Then, it is possible to choose the training algorithm. There are different training algorithms:

• Levenberg-Marquardt: tipically recquires more memory but is usually fastest.

• Bayesian Regularization: takes longer but may be better for challenging problems.

• Scaled Conjugated Gradient: uses less memory. Suitable in low memory situations.

For this example, the Levenberg-Marquardt algorithm is selected, this step is shown in Figure 29.


Neural Time Series (ntstool)

Train Network
Train the network to fit the inputs and targets.

Train Network
Choose a training algorithm: Levenberg-Marquardt

This algorithm typically requires more memory but less time. Training automatically stops when generalization stops improving, as indicated by an increase in the mean square error of the validation samples.

Train using Levenberg-Marquardt. (trainlm)

Results	Target Values	MSE	R
Training:	4598	-	-
Validation:	986	-	-
Testing:	986	-	-

Notes

Training multiple times will generate different results due to different initial conditions and sampling.

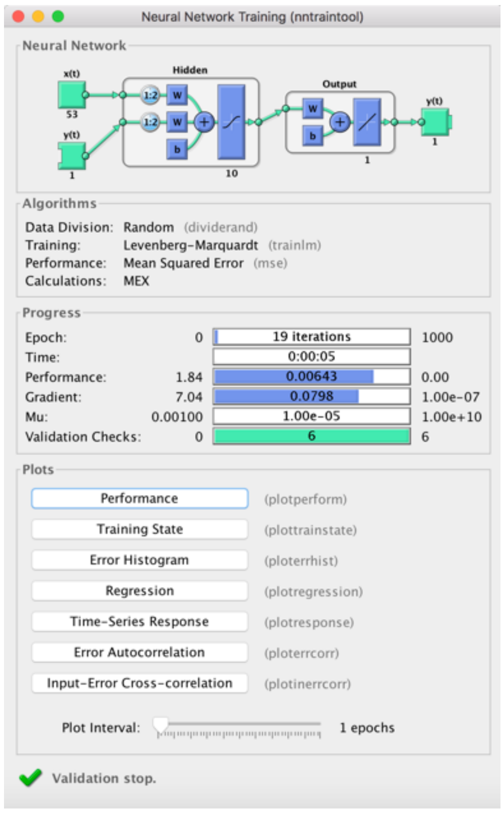
Mean Squared Error is the average squared difference between outputs and targets. Lower values are better. Zero means no error.

Regression R Values measure the correlation between outputs and targets. An R value of 1 means a close relationship, 0 a random relationship.

Train network, then click [Next].

FIGURE 29: CHOOSING TRAINING ALGORITHM

In addition, the tool in Figure 30 provides different plots about the results. One interesting plot is the "Error autocorrelation", shown in Figure 31, which is used to validate the network performance. With a perfect performance, the plot would show only one non-zero value of the autocorrelation, in the 0-Lag position.



Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division:	Random (dividerand)
Training:	Levenberg-Marquardt (trainlm)
Performance:	Mean Squared Error (mse)
Calculations:	MEX

Progress

Epoch:	0	19 iterations	1000
Time:		0:00:05	
Performance:	1.84	0.00643	0.00
Gradient:	7.04	0.0798	1.00e-07
Mu:	0.00100	1.00e-05	1.00e+10
Validation Checks:	0	6	6

Plots

Performance	(plotperform)
Training State	(plottrainstate)
Error Histogram	(ploterrhist)
Regression	(plotregression)
Time-Series Response	(plotresponse)
Error Autocorrelation	(ploterrcorr)
Input-Error Cross-correlation	(plotinerrcorr)

Plot Interval: 1 epochs

✓ Validation stop.

FIGURE 30: TRAINING TOOL

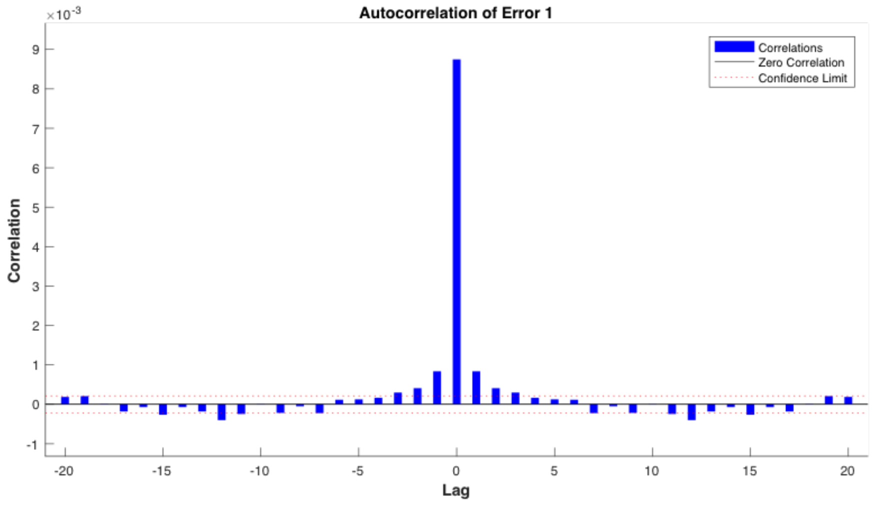


FIGURE 31: AUTOCORRELATION OF ERROR

Once the model is trained, the next step is to generate the MATLAB code to test the model with new data and make predictions. There are three possible configurations to obtain data with this model.

The first one, the standard open loop Neural Network (NN), shown in Figure 32. In this configuration the predicted output data is not connected to the input.

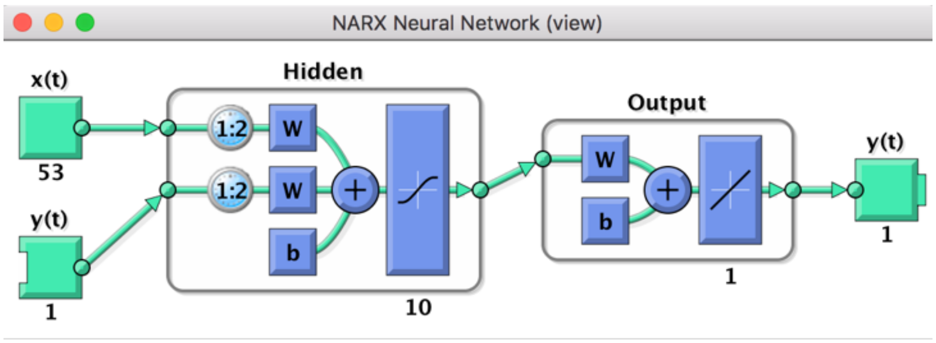


FIGURE 32: FIRST MODEL, FIRST CONFIGURATION: NARX NEURAL NETWORK OPEN LOOP

In Figure 33 the results comparing predicted vs real data for 200 hours using open loop Neural Network configuration are shown.

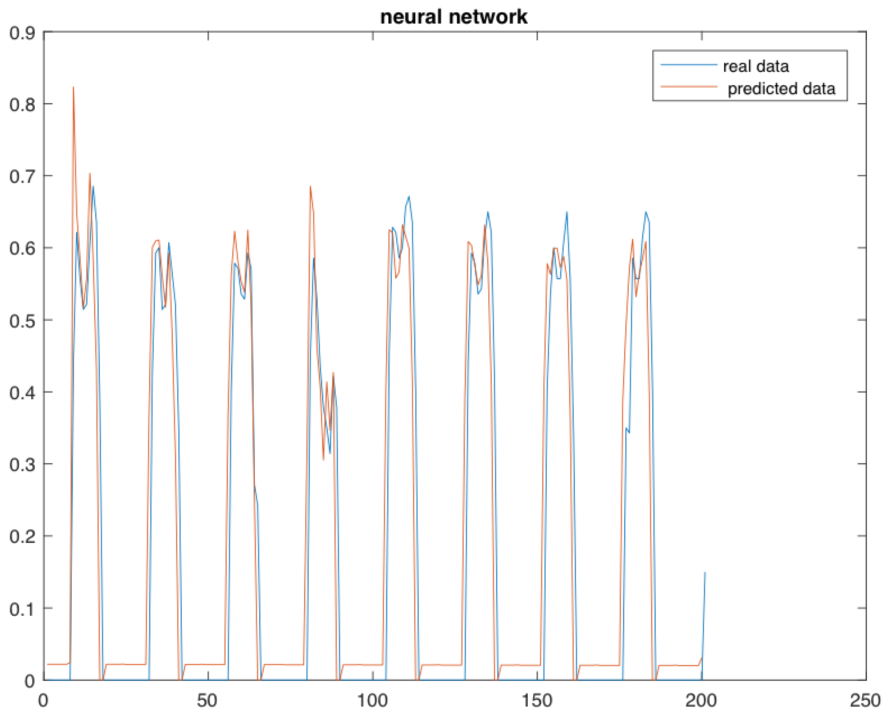


FIGURE 33: PREDICTED DATA VS. REAL DATA FOR FIRST MODEL, FIRST CONFIGURATION: NARX NEURAL NETWORK OPEN LOOP

The second one, Closed Loop Neural Network (CL), shown in Figure 34. In this configuration the predicted output data is connected to the input.

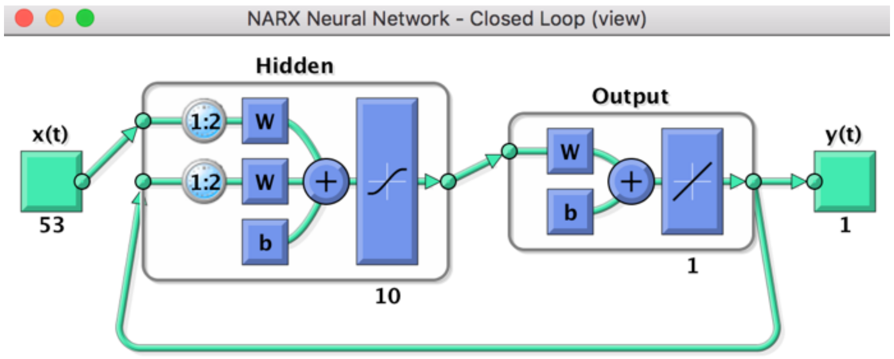


FIGURE 34: FIRST MODEL, SECOND CONFIGURATION: NARX NEURAL NETWORK CLOSED LOOP

In Figure 35 the results of predited vs real data for 200 hours using closed loop Neural Network configuration are shown.

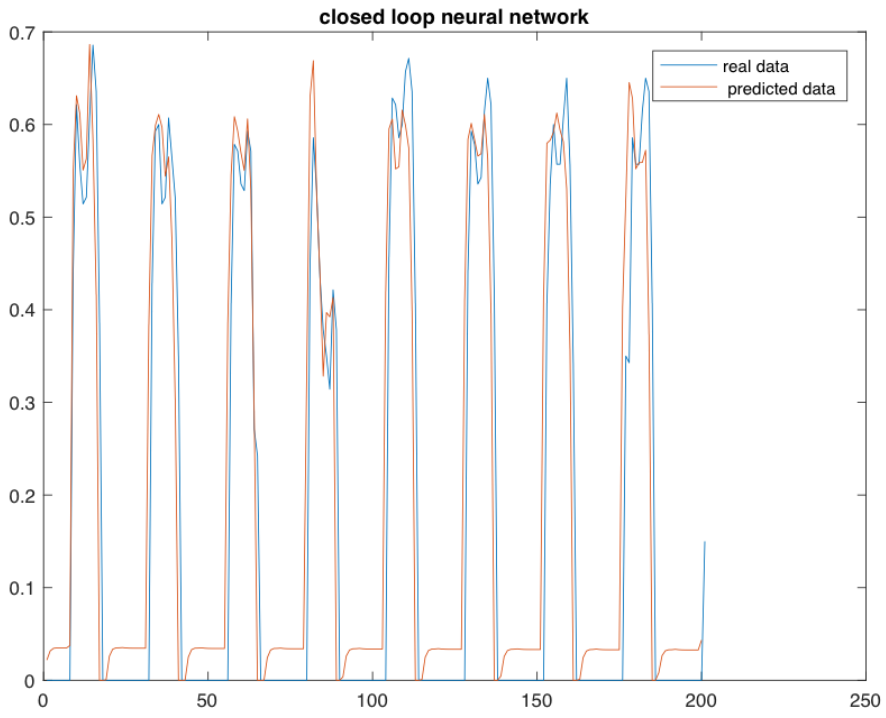


FIGURE 35: PREDICTED DATA VS. REAL DATA FOR FIRST MODEL, SECOND CONFIGURATION: NARX NEURAL NETWORK CLOSED LOOP

The third configuration, Step Ahead Prediction Network (SA), is shown in Figure 36. In this configuration the model predicts data one step ahead using data from the last step, as it only have one step of delay.

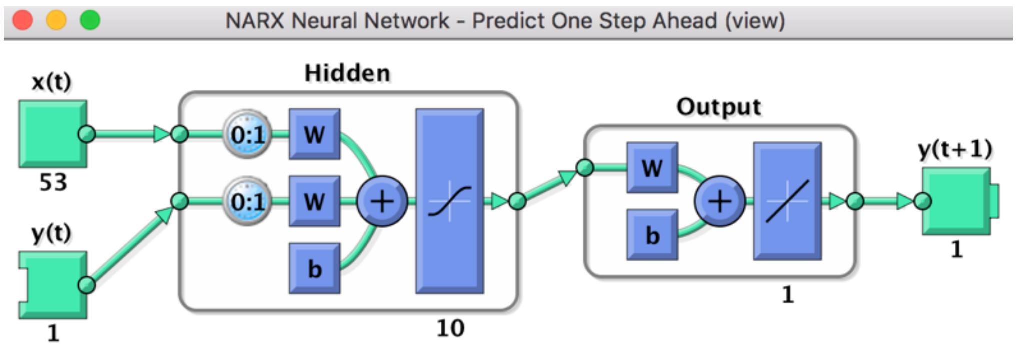


FIGURE 36: FIRST MODEL, THIRD CONFIGURATION: NARX PREDICT ONE STEAP AHEAD

In Fig.37 the results of One Step Ahead configuration are shown.

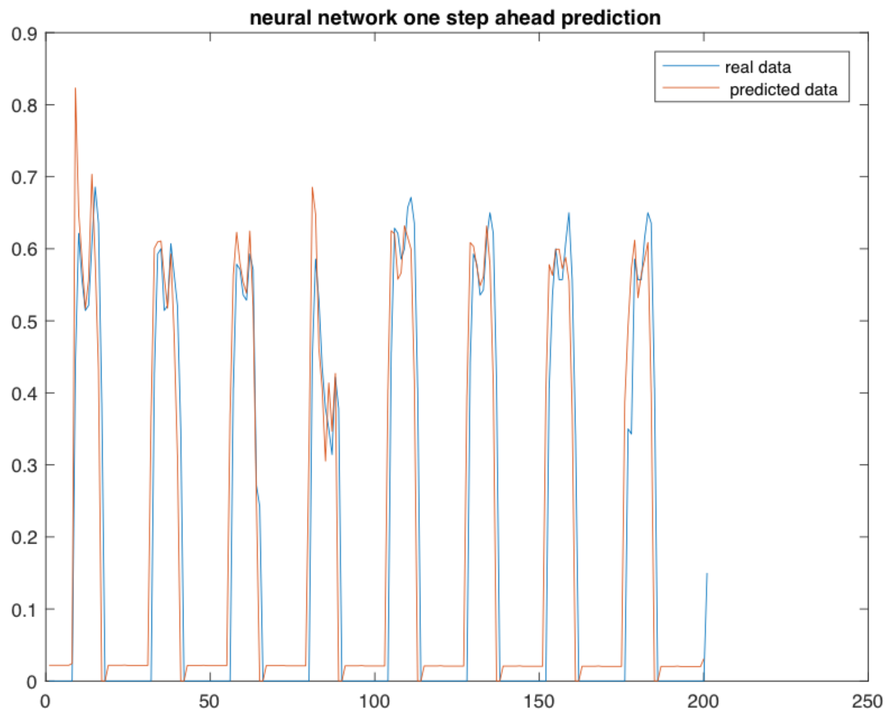


FIGURE 37: PREDICTED DATA VS REAL DATA FOR FIRST MODEL, THIRD CONFIGURATION: NARX PREDICT ONE STEP AHEAD

The models have been tested with different training algorithms and different number of delays. The final results of the RMSE of the models performance are shown in Table 4, and the RMSE of the predicted vs. real data are shown in Table 5.

TABLE 4: RMSE MODEL PERFORMANCE, ANN-NARX

	Algorithm 1 (LM)			Algorithm 2 (BR)			Algorithm 3(SCG)		
	NN	CL	SA	NN	CL	SA	NN	CL	SA
d=2	0.0082	0.0108	0.0082	0.0062	0.0120	0.0062	0.0110	0.0131	0.0110
d=15	0.0278	0.0428	0.0278	______	______	______	0.0077	0.0088	0.0077

Table 5: RMSE predicted vs real data, ANN NARX

	Algorithm 1 (LM)			Algorithm 2 (BR)			Algorithm 3(SCG)		
	NN	CL	SA	NN	CL	SA	NN	CL	SA
d=2	0.0171	0.0187	0.0171	0.0210	0.0245	0.0210	0.0157	0.0186	0.0157
d=15	0.0264	0.0440	0.0264	_____	_____	_____	0.0187	0.0218	0.0187

The results will be compared to the results in Linear Regression and Support Vector Machines Regression in Section 4: Discussion.

Second ANN model: prediction using historical data from the same location

The second model of Artificial Neural Network is the model NAR (Nonlinear Autoregressive). In this case, the model will predict data in one location, using only historical data from that location. The procedure is the same than in NARX model. The NAR model is built and trained with training data and then is tested in the MATLAB code with the testing data.

There are three possible configurations as in the previous section. The first configuration is open loop Neural Network, shown in Figure 38. The difference between NAR and NARX is that NAR only have one input, and NARX had two inputs.

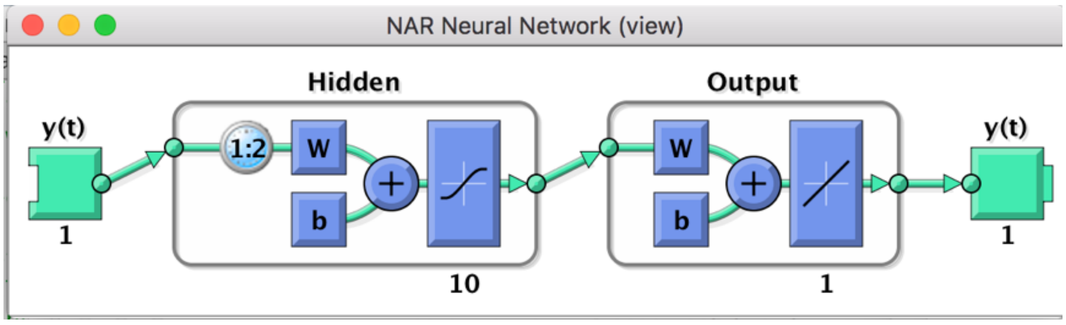


Figure 38: Second model, first configuration: NAR Neural Network open loop

The graphic in Figure 39 shows the results of predicted vs real data using this configuration.

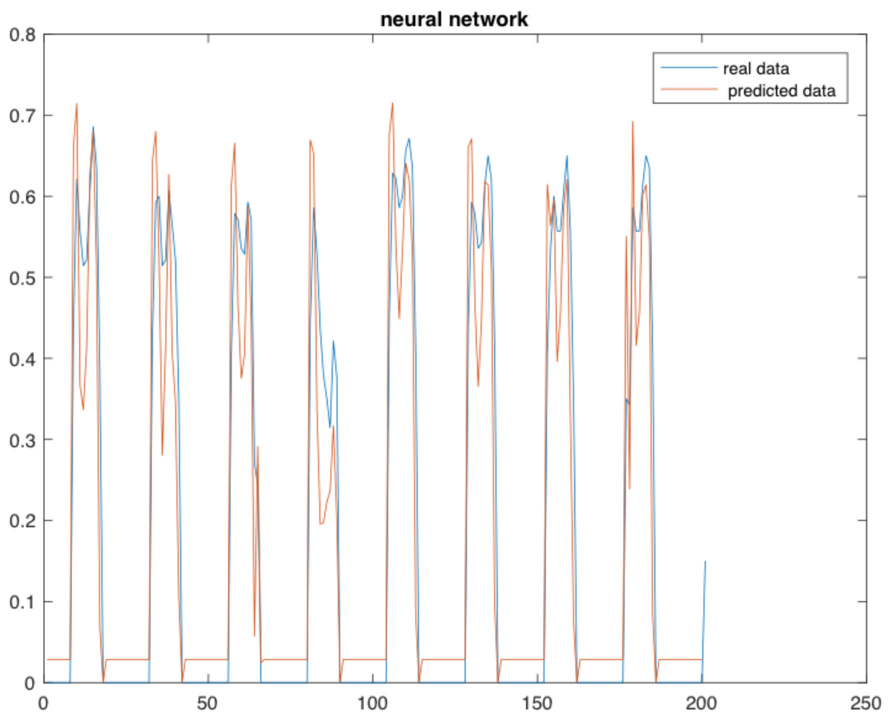


FIGURE 39: PREDICTED DATA VS. REAL DATA FOR SECOND MODEL, FIRST CONFIGURATION: NAR NEURAL NETWORK OPEN LOOP

The second configuration is closed loop Neural Network, see in Fig. 40.

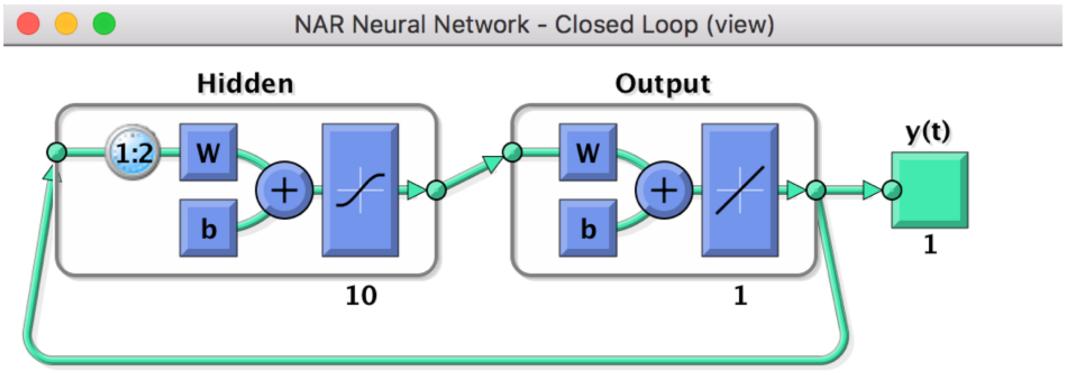


FIGURE 40: SECOND MODEL, SECOND CONFIGURATION: NAR NEURAL NETWORK CLOSED LOOP

And the Figure 41 shows the results of predicted vs real data using closed loop configuration.

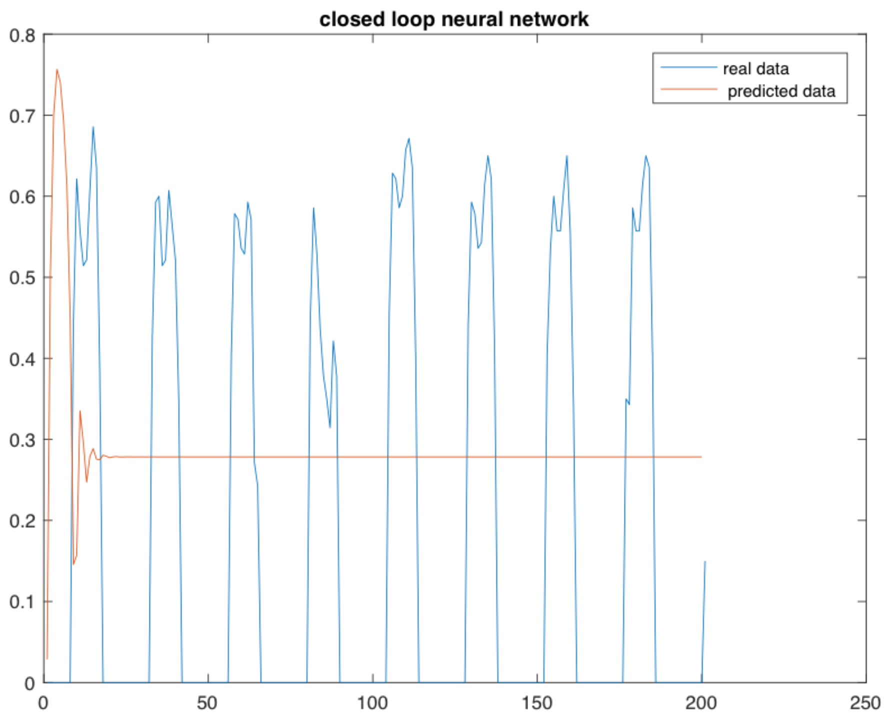


FIGURE 41: PREDICTED DATA VS. REAL DATA FOR SECOND MODEL, SECOND CONFIGURATION: NAR NEURAL NETWORK CLOSED LOOP

The third configuration is Predict One Step Ahead Neural Network, shown in Figure 42.

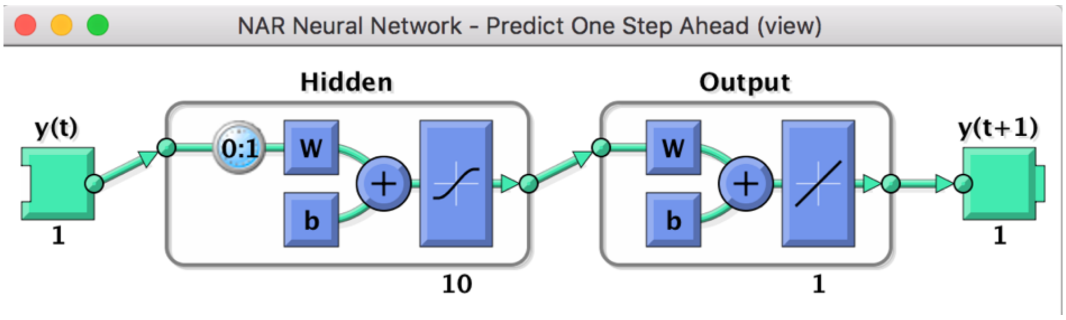


FIGURE 42: SECOND MODEL, THIRD CONFIGURATION: NAR PREDICT ONE STEAP AHEAD

The graphic in Figure 43 shows the results of predicted vs real data using this configuration.

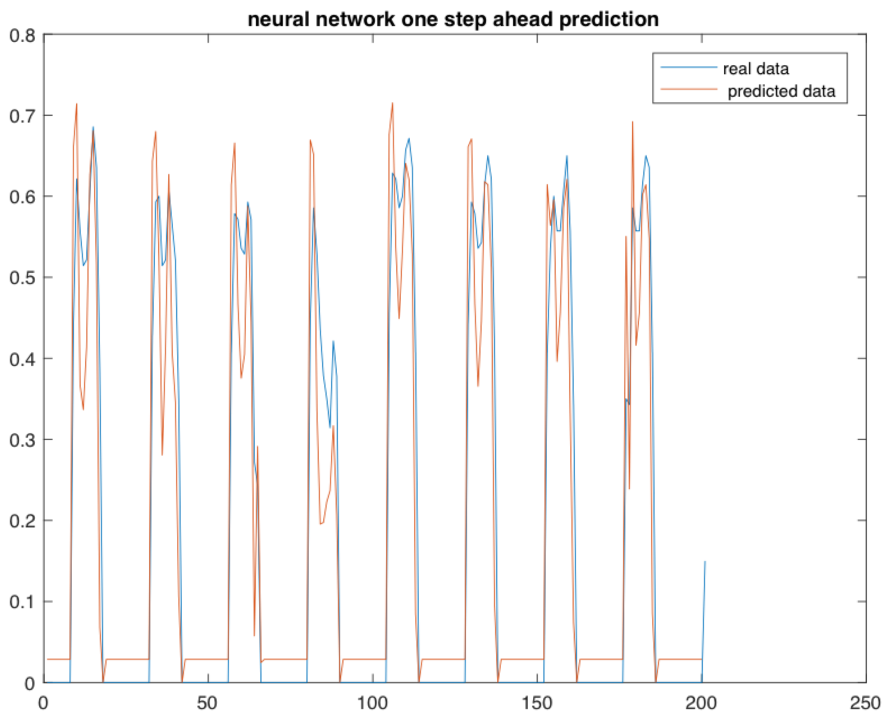


FIGURE 43: PREDICTED DATA VS REAL DATA FOR SECOND MODEL, THIRD CONFIGURATION: NARX PREDICT ONE STEP AHEAD

The models have been tested with the different training algorithms: Levenberg-Marquardt (LM), Bayesian Regularization (BR) and Scaled Conjugated Gradient (SCG), and different number of delays, d=2 and d=15. The final results of the RMSE of the models performance are shown in Table 6, and the RMSE of the predicted vs. real data are shown in Table 7. For each algorithm it is shown the results of every configuration, Open Loop Neural Network (NN), Closed Loop Neural Network (CL), and One Step Ahead Prediction (SA).

TABLE 6: RMSE MODEL PERFORMANCE, ANN-NAR

	Algorithm 1 (LM)			Algorithm 2 (BR)			Algorithm 3(SCG)		
	NN	CL	SA	NN	CL	SA	NN	CL	SA
d=2	0.0161	0.1277	0.0161	0.0161	14.1298	0.0161	0.0198	0.1304	0.0198
d=15	0.0107	0.9252	0.0107	0.0105	0.1966	0.0105	0.0136	0.1856	0.0136

TABLE 7: RMSE PREDICTED VS REAL DATA, ANN - NAR

	Algorithm 1 (LM)			Algorithm 2 (BR)			Algorithm 3(SCG)		
	NN	CL	SA	NN	CL	SA	NN	CL	SA
d=2	0.0100	0.1254	0.0100	0.0093	0.2187	0.0094	0.0056	0.1303	0.0056
d=15	0.0160	0.7347	0.0160	0.0155	0.1948	0.0155	0.0122	0.1827	0.0122

3.3.4 Changes in the dataset

As the results were not as accurate as it was expected, a change in the dataset is done. For the next tests, the month and day of the month will not be used as predictors, as this type of data can be affecting to the predictions. The results are the following:

In the case of Linear Regression using data from nearest locations, there is no change, as the date was not used as a predictor. In the case of Linear Regression using data from every location, there is no change in the results neither.

In the case of Support Vector Machines, using data from nearest locations, there is no change as the date was not used initially as a predictor, and using data from every location, the results did not significantly changed. The results are shown in Table 8, the values with an asterisk are the values obtained changing the dataset without the date as a predictor.

TABLE 8: RMSE, SVMR MODEL, WITH DATE AND WITHOUT DATE AS A PREDICTOR(VALUES WITH ASTERISK)

	SVM linear kernel, with data from nearest locations	SVM linear kernel, with data from every location.
Model	0.2370	0.0664 *0.0778
Predicted/real data	0.0792	0.0026 *0.0025

In the case of Artificial Neural Network - NARX model, the results without date as a predictor, are slightly better in every algorithm. The greatest improvement is in the algorithm LM with a delay of d=15, for the model performance. These results are shown in the Tables 9 and 10.

In the case of Artifical Neural Network- NAR model, there are no changes in the results, as the date was not used as a predictor, only historical data from the location to predict.

TABLE 9: RMSE MODEL PERFORMANCE, ANN-NARX, WITH DATE AND WITHOUT DATE AS A PREDICTOR(VALUES WITH ASTERISK)

	Algorithm 1 (LM)			Algorithm 2 (BR)			Algorithm 3(SCG)		
	NN	CL	SA	NN	CL	SA	NN	CL	SA
d=2	0.0082	0.0108	0.0082	0.0062	0.0120	0.0062	0.0110	0.0131	0.0110

	*0.0091	*0.0147	*0.0091	*0.0063	*0.0091	*0.0063	*0.0100	*0.0130	*0.0100
d=15	0.0278 *0.0087	0.0428 *0.0129	0.0278 *0.0087	_____	_____	_____	0.0077 *0.0076	0.0088 *0.0093	0.0077 *0.0076

Table 10: RMSE predicted vs real data, ANN-NARX with date and without date as a predictor(values with asterisk)

	Algorithm 1 (LM)			Algorithm 2 (BR)			Algorithm 3(SCG)		
	NN	CL	SA	NN	CL	SA	NN	CL	SA
d=2	0.0171 *0.0161	0.0187 *0.0254	0.0171 *0.0161	0.0210 *0.0201	0.0245 *0.0221	0.0210 *0.0202	0.0157 *0.0149	0.0186 *0.0180	0.0157 *0.0150
d=15	0.0264 *0.0167	0.0440 *0.0207	0.0264 *0.0167	_____	_____	_____	0.0187 *0.0192	0.0218 *0.0235	0.0187 *0.0192

4 Discussion

This chapter is focused on discuss the results of the experiments carried out in the previous chapter.

On the first place, the results of the first Machine Learning technique tested in this thesis, Linear Regression, show that there is a better prediction of data in one location, if historical data from every location is used as a predictor, instead of the case of using only data from nearest locations. This can be due to the fact of using a not large enough dataset, with historical data from only one year.

The best results provide by Linear Regression are: RMSE model=0.0610 and RMSE predicted/real=0.0016.

On the second place, in the case of Support Vector Machines Regression, the smaller RMSE is also obtained in the case of prediction using data from every location, using the model with a linear kernel. Using quadratic and cubic kernel did not provide good results. Regarding to the model, the best RMSE obtained was RMSE model=0.0664 and regarding to the comparison of predicted vs. real data in the testing stage, the best result was RMSE predicted/real=0.0026. However, this error is not smaller than the one ob- tained in the Linear Regression model.

By contrast, in the third technique, Artificial Neural Networks, the configuration that provides the smaller RMSE comparing predicted vs real data is the NAR model, which predicts data using historical data from the same location, using the algorithm Scaled Conjugated Gradient (SCG), and a delay d=2. The error obtained was RMSE predicted/real= 0.0056. Regarding to the model performance, the NARX model with the algorithm Bayesian Regularization (BR) and a delay of d=2, provides the best results in this technique, with an error of RMSE model=0.0062. Some results in ANN could be improved trying different number of hidden neurons or different number of delays.

In addition, it is noteworthy that in the case of ANN, the configuration Closed Loop Neural Network, did not work correctly making predictions, and for a delay d=15 in the NARX model, the algorithm BR was not able to be trained, due to it required an excessive amount of time.

With regarding to the results using the dataset without the columns of the month and day of the month, in the Section 4.3.4, there was not the expected improvement in the results, only in the case of the NARX model of ANN the results slightly improved.

The errors in the predictions can also be due to the dataset was divided into training and testing data with a ratio of 75/25, i.e, the training data has values from January to October, and the testing data has values from October to December, which may affect to the accuracy of the results.

Comparing these results to the results obained in [14], which is a study of predicting daily mean solar power using machine learning techniques, the results in [14] were more accurate than the results obtained in this thesis, since not only historical data but more parameters are used as a predictors, such as solar irradiance and solar angles azimuth and zenith, and the technique that provided the best results was Artificial Neural Network, in contrast to this thesis where the best technique is Linear Regression.

Moreover, in another related article, [15], where some machine learning techniques are implemented to predict wind power, the predictive models are obtained using different software, in that case using the tools and libraries provided by Python. In contrast to this thesis, in [15] the best results were provided by the algorithm Support Vector Machines.

To sum up, the Machine Learning algorithm with the best performance in making prediction of solar power data is Linear Regression, using historical data from every location of the dataset.

5 Conclusions

In this project, some Machine Learning techniques have been tested with the aim to make predictions of solar energy data. The results show that, despite the wide variety of techniques and complex algorithms, which could be improved using a different dataset or adding weather features in order to make more accurate predictions, one of the most simple techniques, Linear Regression, have provided the best results in making predictions from historical data.

That means that, it could be possible and not a hard task to implement Machine Learning techniques in photovoltaic plants, in order to forecast the generated power in a solar cell, helping to reduce the waste of electrical energy, meeting the demand of the electric grid in an optimized way, and making renewables energy systems more efficient.

Conclusiones

En este proyecto, se han implementado algunas técnicas de Machine Learning con el objetivo de hacer predicciones de datos de energía solar. Los resultados muestran que, a pesar de la amplia variedad de técnicas y algoritmos más complejos, se podrían mejorar utilizando un dataset diferente o añadiendo diferentes características para hacer predicciones más precisas, una de las técnicas más simples, Regresión Lineal, ha proporcionado los mejores resultados haciendo predicciones con datos históricos.

Esto significa que, podría ser possible y una tarea no muy compleja, implementar técnicas de Machine Learning en plantas fotovoltaicas, para predecir la energía generada en un panel solar, ayudando a reducir el malgasto de energía eléctrica, y cumpliendo la demanda de la red eléctrica de una forma optimizada, haciendo los sistemas de energías renovables más eficientes.

References

[1] Photovoltaics, https://en.wikipedia.org/wiki/Photovoltaics

[2] Global Data, https://energy.globaldata.com/

[3] V. Masson, M. Bonhomme, J.L. Salagnac, X. Briottet and A. Lemonsu, "Solar panels reduce both global warming and urban heat island", 2014, *Front. Environ. Sci.* 2:14. doi: 10.3389/fenvs.2014.00014

[4] United Nations, "Climate Change", http://www.un.org/en/sections/issues-depth/climate-change/

[5] A History of Solar Cells, https://www.solarpowerauthority.com/a- history-of-solar-cells/

[6] Mathworks, "What is Machine Learning?" https://se.mathworks.com/discovery/machine-learning.html

[7] T.M. Mitchell, *Machine Learning*. McGraw-Hill International Editions, 1997.

[8] K. Hemant and C. Rishabh, "Comprehensive Review On Supervised Machine Learning Algorithms". IEEE, International Conference on Machine Learning and Data Science, 2017.

[9] Logistic Function, https://en.wikipedia.org/wiki/Logistic_function

[10] SVM Theory, https://medium.com/machine-learning-101/chapter- 2-svm-support-vector-machine-theory-f0812effc72

[11] National Renewable Energy Laboratory (NREL), "Solar Power Data for Integration Studies", https://www.nrel.gov/grid/solar-power-data.html

[12] Mathworks, "Statistics and Machine Learning Toolbox, https://se.mathworks.com/products/statistics.html

[13] Mathworks, "Neural Network Toolbox", https://se.mathworks.com/products/neural-network.html

[14] F. Jawaid and K. Nazirjunejo, "Predicting Daily Mean Solar Power Using Machine Learning Regression Techniques". IEEE, The Sixth International Conference on Innovative Computing Technology (INTECH 2016)

[15] J. Lässig, K. Kersting, and K. Morik, *Computational Sustainability*, Springer Intenrational Publishing Switzerland, 2016. Chapter 2: “Wind Power Prediction with Machine Learning”, N.A. Treiber, J. Heinermann and O. Kramer.

Appendix A

Here is the MATLAB code and functions used in the first tested algorithm: Linear Regression.

A.1 LR First Model

```
%FIRST MODEL LINEAR REGRESSION
%this model calculate predictions for location 52, using info from
%locations 50 51 53 and 54 (nearest locations)

clear all
close all

load finaldataset.csv
data=finaldataset;
%first column=month
%second column = day of the month
%rest of columns = solar energy in different locations (52)

%normalization of the values from 0 to 1
data_norm=zeros(8760,54);
for i=3:54
data_norm(:,i)=(data(:,i)-min(data(:,i)))/(max(data(:,i))-
min(data(:,i)));
end
data_norm(:,1:2)=data(:,1:2);

% dividing data into training and testing data
% ratio 75/25

training_data=data_norm(1:0.75*length(data_norm),:);

test_data=data_norm(0.75*length(data_norm):length(data_norm),:);



t2=[test_data(:,50), test_data(:,51), test_data(:,53),
test_data(:,54)];


%train the model
[trainedModel_tree, validationRMSE_tree] =
trainRegressionModel_tree(training_data)
%make predictions with test data
yfit = trainedModel_tree.predictFcn(t2);

%creating plots to compare predicted data vs real data
figure
plot(test_data(1:100,52))  %100 hours of real data of column 52
title('Linear Regression- first model')

hold on

plot(yfit(1:100)) %100 hours of predicted data in column 52
xlabel('first 100 hours ')
ylabel('Solar Energy')
legend('real data','predicted data')
```

```
%plot 1 month
figure
plot(test_data(1:720,52))
title('Linear Regression - first model')

hold on

plot(yfit(1:720))
xlabel('1 month ')
ylabel('Solar Energy')
legend('real data','predicted data')

%calculate Root Mean Squared Error between predicted and real data
err_LR=immse(test_data(:,52),yfit)
```

A.1 LR First Model - function training model

```
function [trainedModel_tree, validationRMSE_tree] =
trainRegressionModel_tree(trainingData)
% [trainedModel, validationRMSE] =
trainRegressionModel(trainingData)
% returns a trained regression model and its RMSE. This code
recreates the
% model trained in Regression Learner app. Use the generated code
to
% automate training the same model with new data, or to learn how
to
% programmatically train models.
%
%  Input:
%      trainingData: a matrix with the same number of columns and
data type
%       as imported into the app.
%
%  Output:
%      trainedModel: a struct containing the trained regression
model. The
%       struct contains various fields with information about the
trained
%       model.
%
%      trainedModel.predictFcn: a function to make predictions on
new data.
%
%      validationRMSE: a double containing the RMSE. In the app,
the
%       History list displays the RMSE for each model.
%
% Use the code to train the model with new data. To retrain your
model,
% call the function from the command line with your original data
or new
% data as the input argument trainingData.
%
% For example, to retrain a regression model trained with the
original data
% set T, enter:
%   [trainedModel, validationRMSE] = trainRegressionModel(T)
%
% To make predictions with the returned 'trainedModel' on new data
T2, use
%   yfit = trainedModel.predictFcn(T2)
%
% T2 must be a matrix containing only the predictor columns used
for
% training. For details, enter:
%   trainedModel.HowToPredict

% Auto-generated by MATLAB on 16-May-2018 23:56:43


% Extract predictors and response
% This code processes the data into the right shape for training
the
% model.
% Convert input to table
```

```
inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9', 'column_10',
'column_11', 'column_12', 'column_13', 'column_14', 'column_15',
'column_16', 'column_17', 'column_18', 'column_19', 'column_20',
'column_21', 'column_22', 'column_23', 'column_24', 'column_25',
'column_26', 'column_27', 'column_28', 'column_29', 'column_30',
'column_31', 'column_32', 'column_33', 'column_34', 'column_35',
'column_36', 'column_37', 'column_38', 'column_39', 'column_40',
'column_41', 'column_42', 'column_43', 'column_44', 'column_45',
'column_46', 'column_47', 'column_48', 'column_49', 'column_50',
'column_51', 'column_52', 'column_53', 'column_54'});

predictorNames = {'column_50', 'column_51', 'column_53',
'column_54'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_52;
isCategoricalPredictor = [false, false, false, false];

% Train a regression model
% This code specifies all the model options and trains the model.
regressionTree = fitrtree(...
    predictors, ...
    response, ...
    'MinLeafSize', 12, ...
    'Surrogate', 'off');

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames',
predictorNames);
treePredictFcn = @(x) predict(regressionTree, x);
trainedModel_tree.predictFcn = @(x)
treePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedModel_tree.RegressionTree = regressionTree;
trainedModel_tree.About = 'This struct is a trained model exported
from Regression Learner R2017b.';
trainedModel_tree.HowToPredict = sprintf('To make predictions on a
new predictor column matrix, X, use: \n  yfit = c.predictFcn(X)
\nreplacing ''c'' with the name of the variable that is this
struct, e.g. ''trainedModel''. \n \nX must contain exactly 4
columns because this model was trained using 4 predictors. \nX
must contain only predictor columns in exactly the same order and
format as your training \ndata. Do not include the response column
or any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appregression_exportmodeltoworkspace'')">How to predict using an
exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training
the
% model.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9', 'column_10',
'column_11', 'column_12', 'column_13', 'column_14', 'column_15',
'column_16', 'column_17', 'column_18', 'column_19', 'column_20',
'column_21', 'column_22', 'column_23', 'column_24', 'column_25',
'column_26', 'column_27', 'column_28', 'column_29', 'column_30',
'column_31', 'column_32', 'column_33', 'column_34', 'column_35',
'column_36', 'column_37', 'column_38', 'column_39', 'column_40',
'column_41', 'column_42', 'column_43', 'column_44', 'column_45',
'column_46', 'column_47', 'column_48', 'column_49', 'column_50',
'column_51', 'column_52', 'column_53', 'column_54'});

predictorNames = {'column_50', 'column_51', 'column_53',
'column_54'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_52;
isCategoricalPredictor = [false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedModel_tree.RegressionTree,
'KFold', 5);

% Compute validation predictions
validationPredictions = kfoldPredict(partitionedModel);

% Compute validation RMSE
validationRMSE_tree = sqrt(kfoldLoss(partitionedModel, 'LossFun',
'mse'));
```

A.2 LR Second Model

```
%SECOND MODEL LINEAR REGRESSION

%this model calculate predictions for location 52, using info from
%every location of the dataset

clear all
close all

load finaldataset.csv
data=finaldataset;
%first column=month
%second column = day of the month
%rest of columns = solar energy in different locations (52)



%normalization of the values from 0 to 1
data_norm=zeros(8760,54);
for i=3:54
data_norm(:,i)=(data(:,i)-min(data(:,i)))/(max(data(:,i))-
min(data(:,i)));
end

data_norm(:,1:2)=data(:,1:2);


% dividing data into training and testing data
% ratio 75/25

training_data=data_norm(1:0.75*length(data_norm),:);

test_data=data_norm(0.75*length(data_norm):length(data_norm),:);


t2=[test_data(:,1:51), test_data(:,53:54)];


%train the model
[trainedModel_LR, validationRMSE_LR] =
trainRegressionModel_LR(training_data)

%make predictions with test data
yfit_LR= trainedModel_LR.predictFcn(t2);

%creating plots to compare predicted data vs real data
figure
plot(test_data(1:100,52))
title('Linear Regression - second model')

hold on

plot(yfit_LR(1:100)) %100 hours of real data of column 52
xlabel('first 100 hours ')
ylabel('Solar Energy')
legend('real data','predicted data')
```

```
%plot 1 month
figure
plot(test_data(1:720,52))
title('Linear Regression - second model')

hold on

plot(yfit_LR(1:720))
xlabel('1 month ')
ylabel('Solar Energy')
legend('real data','predicted data')

%calculate Root Mean Squared Error between predicted and real data
err_LR=immse(test_data(:,52),yfit_LR)
```

A.2 LR Second Model- function training model

```
 function [trainedModel_LR, validationRMSE_LR] =
trainRegressionModel_LR(trainingData)

inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9', 'column_10',
'column_11', 'column_12', 'column_13', 'column_14', 'column_15',
'column_16', 'column_17', 'column_18', 'column_19', 'column_20',
'column_21', 'column_22', 'column_23', 'column_24', 'column_25',
'column_26', 'column_27', 'column_28', 'column_29', 'column_30',
'column_31', 'column_32', 'column_33', 'column_34', 'column_35',
'column_36', 'column_37', 'column_38', 'column_39', 'column_40',
'column_41', 'column_42', 'column_43', 'column_44', 'column_45',
'column_46', 'column_47', 'column_48', 'column_49', 'column_50',
'column_51', 'column_52', 'column_53', 'column_54'});

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4',
'column_5', 'column_6', 'column_7', 'column_8', 'column_9',
'column_10', 'column_11', 'column_12', 'column_13', 'column_14',
'column_15', 'column_16', 'column_17', 'column_18', 'column_19',
'column_20', 'column_21', 'column_22', 'column_23', 'column_24',
'column_25', 'column_26', 'column_27', 'column_28', 'column_29',
'column_30', 'column_31', 'column_32', 'column_33', 'column_34',
'column_35', 'column_36', 'column_37', 'column_38', 'column_39',
'column_40', 'column_41', 'column_42', 'column_43', 'column_44',
'column_45', 'column_46', 'column_47', 'column_48', 'column_49',
'column_50', 'column_51', 'column_53', 'column_54'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_52;
isCategoricalPredictor = [false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false];

% Train a regression model
% This code specifies all the model options and trains the model.
concatenatedPredictorsAndResponse = predictors;
concatenatedPredictorsAndResponse.column_52 = response;
linearModel = fitlm(...
    concatenatedPredictorsAndResponse, ...
    'linear', ...
    'RobustOpts', 'off');

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames',
predictorNames);
linearModelPredictFcn = @(x) predict(linearModel, x);
trainedModel_LR.predictFcn = @(x)
linearModelPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedModel_LR.LinearModel = linearModel;
trainedModel_LR.About = 'This struct is a trained model exported
from Regression Learner R2017b.';
```

```
trainedModel_LR.HowToPredict = sprintf('To make predictions on a
new predictor column matrix, X, use: \n  yfit = c.predictFcn(X)
\nreplacing ''c'' with the name of the variable that is this
struct, e.g. ''trainedModel''. \n \nX must contain exactly 53
columns because this model was trained using 53 predictors. \nX
must contain only predictor columns in exactly the same order and
format as your training \ndata. Do not include the response column
or any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appregression_exportmodeltoworkspace'')">How to predict using an
exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training
the
% model.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9', 'column_10',
'column_11', 'column_12', 'column_13', 'column_14', 'column_15',
'column_16', 'column_17', 'column_18', 'column_19', 'column_20',
'column_21', 'column_22', 'column_23', 'column_24', 'column_25',
'column_26', 'column_27', 'column_28', 'column_29', 'column_30',
'column_31', 'column_32', 'column_33', 'column_34', 'column_35',
'column_36', 'column_37', 'column_38', 'column_39', 'column_40',
'column_41', 'column_42', 'column_43', 'column_44', 'column_45',
'column_46', 'column_47', 'column_48', 'column_49', 'column_50',
'column_51', 'column_52', 'column_53', 'column_54'});

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4',
'column_5', 'column_6', 'column_7', 'column_8', 'column_9',
'column_10', 'column_11', 'column_12', 'column_13', 'column_14',
'column_15', 'column_16', 'column_17', 'column_18', 'column_19',
'column_20', 'column_21', 'column_22', 'column_23', 'column_24',
'column_25', 'column_26', 'column_27', 'column_28', 'column_29',
'column_30', 'column_31', 'column_32', 'column_33', 'column_34',
'column_35', 'column_36', 'column_37', 'column_38', 'column_39',
'column_40', 'column_41', 'column_42', 'column_43', 'column_44',
'column_45', 'column_46', 'column_47', 'column_48', 'column_49',
'column_50', 'column_51', 'column_53', 'column_54'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_52;
isCategoricalPredictor = [false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false];

% Perform cross-validation
KFolds = 5;
cvp = cvpartition(size(response, 1), 'KFold', KFolds);
% Initialize the predictions to the proper sizes
validationPredictions = response;
for fold = 1:KFolds
    trainingPredictors = predictors(cvp.training(fold), :);
    trainingResponse = response(cvp.training(fold), :);
    foldIsCategoricalPredictor = isCategoricalPredictor;
    % Train a regression model
    % This code specifies all the model options and trains the
model.
    concatenatedPredictorsAndResponse = trainingPredictors;
    concatenatedPredictorsAndResponse.column_52 =
trainingResponse;
    linearModel = fitlm(...
        concatenatedPredictorsAndResponse, ...
        'linear', ...
        'RobustOpts', 'off');

    % Create the result struct with predict function
    linearModelPredictFcn = @(x) predict(linearModel, x);
    validationPredictFcn = @(x) linearModelPredictFcn(x);

    % Add additional fields to the result struct

    % Compute validation predictions
    validationPredictors = predictors(cvp.test(fold), :);
    foldPredictions = validationPredictFcn(validationPredictors);

    % Store predictions in the original order
    validationPredictions(cvp.test(fold), :) = foldPredictions;
end

% Compute validation RMSE
isNotMissing = ~isnan(validationPredictions) & ~isnan(response);
validationRMSE_LR = sqrt(nansum(( validationPredictions - response
).^2) / numel(response(isNotMissing) ));
```

Appendix B

Here is the MATLAB code and functions used in the second tested algorithm: Support Vector Machines Regression

B.1 SVMR First Model

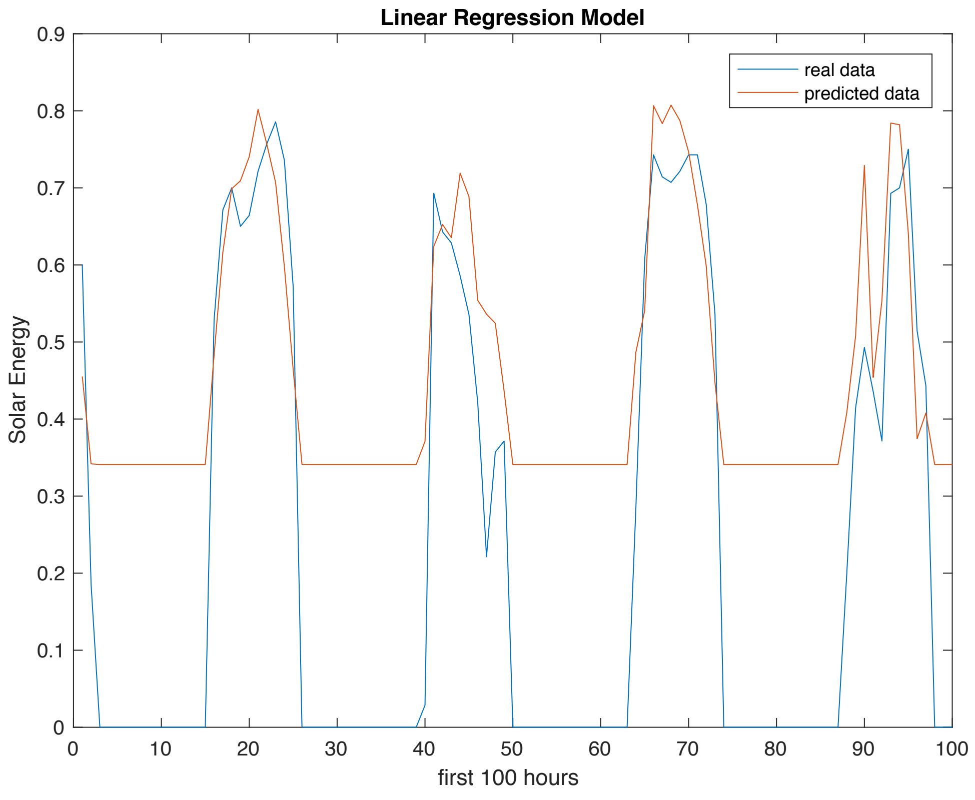
```
%first model SVMR
%this model calculate predictions for location 52, using info from
%locations 50 51 53 and 54 (nearest locations)

clear all
close all

load finaldataset.csv
data=finaldataset;
%first column=month
%second column = day of the month
%rest of columns = solar energy in different locations (52)


%normalization of the values from 0 to 1
data_norm=zeros(8760,54);
for i=3:54
data_norm(:,i)=(data(:,i)-min(data(:,i)))/(max(data(:,i))-
min(data(:,i)));
end

data_norm(:,1:2)=data(:,1:2);


% dividing data into training and testing data
% ratio 75/25

training_data=data_norm(1:0.75*length(data_norm),:);

test_data=data_norm(0.75*length(data_norm):length(data_norm),:);

t2=[test_data(:,50), test_data(:,51), test_data(:,53),
test_data(:,54)];


%train the model
 [trainedModel, validationRMSE] =
trainRegressionModel_SVMR1(training_data)
 %make predictions with test data
 yfit = trainedModel.predictFcn(t2);



%creating plots to compare predicted data vs real data
figure
plot(test_data(1:100,52))%100 hours of real data of column 52
title('SVMR Model')

hold on
```

```
plot(yfit(1:100)) %100 hours of predicted data in column 52
xlabel('first 100 hours ')
ylabel('Solar Energy')
legend('real data','predicted data')

%plot 1 month
figure
plot(test_data(1:720,52))
title('SVMR Model')

hold on

plot(yfit(1:720))
xlabel('1 month ')
ylabel('Solar Energy')
legend('real data','predicted data')


%calculate Root Mean Squared Error between predicted and real data
err_SVM=immse(test_data(:,52),yfit)
```

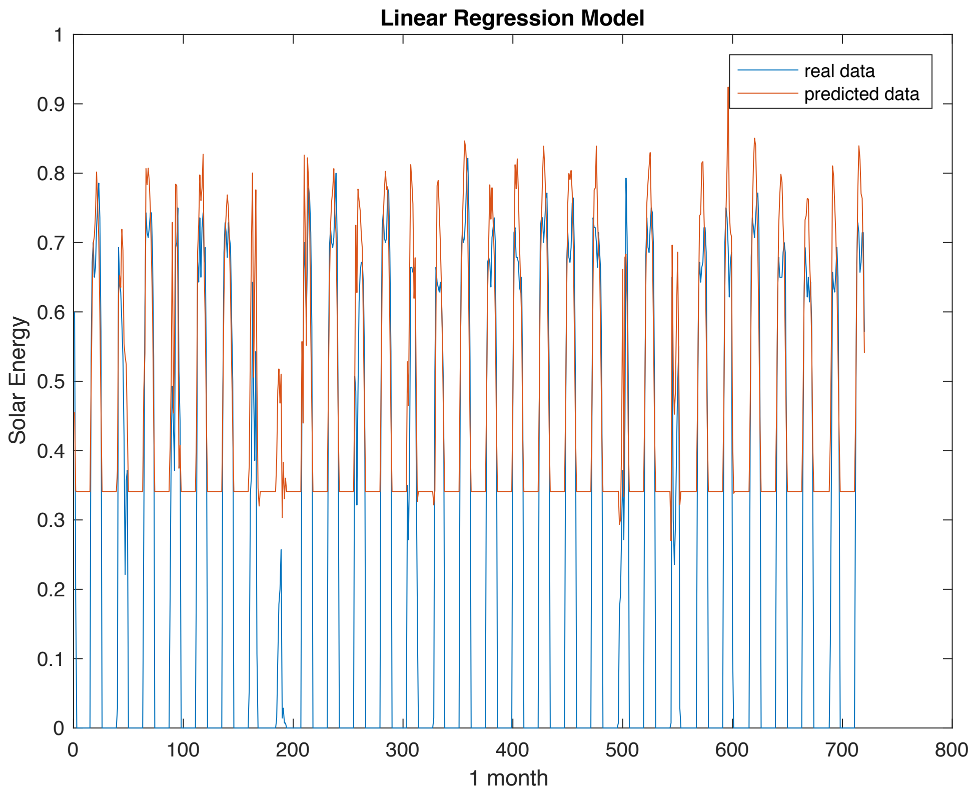
B.1 SVMR First Model - function training model

```
function [trainedModel, validationRMSE] =
trainRegressionModel_SVMR1(trainingData)

inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9', 'column_10',
'column_11', 'column_12', 'column_13', 'column_14', 'column_15',
'column_16', 'column_17', 'column_18', 'column_19', 'column_20',
'column_21', 'column_22', 'column_23', 'column_24', 'column_25',
'column_26', 'column_27', 'column_28', 'column_29', 'column_30',
'column_31', 'column_32', 'column_33', 'column_34', 'column_35',
'column_36', 'column_37', 'column_38', 'column_39', 'column_40',
'column_41', 'column_42', 'column_43', 'column_44', 'column_45',
'column_46', 'column_47', 'column_48', 'column_49', 'column_50',
'column_51', 'column_52', 'column_53', 'column_54'});

predictorNames = {'column_50', 'column_51', 'column_53',
'column_54'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_52;
isCategoricalPredictor = [false, false, false, false];

% Train a regression model
% This code specifies all the model options and trains the model.
responseScale = iqr(response);
if ~isfinite(responseScale) || responseScale == 0.0
    responseScale = 1.0;
end
boxConstraint = responseScale/1.349;
epsilon = responseScale/13.49;
regressionSVM = fitrsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'linear', ...
    'PolynomialOrder', [], ...
    'KernelScale', 'auto', ...
    'BoxConstraint', boxConstraint, ...
    'Epsilon', epsilon, ...
    'Standardize', true);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames',
predictorNames);
svmPredictFcn = @(x) predict(regressionSVM, x);
trainedModel.predictFcn = @(x)
svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedModel.RegressionSVM = regressionSVM;
trainedModel.About = 'This struct is a trained model exported from
Regression Learner R2017b.';
trainedModel.HowToPredict = sprintf('To make predictions on a new
predictor column matrix, X, use: \n  yfit = c.predictFcn(X)
\nreplacing ''c'' with the name of the variable that is this
struct, e.g. ''trainedModel''. \n \nX must contain exactly 4
columns because this model was trained using 4 predictors. \nX
must contain only predictor columns in exactly the same order and
format as your training \ndata. Do not include the response column
or any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appregression_exportmodeltoworkspace'')">How to predict using an
exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training
the
% model.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9', 'column_10',
'column_11', 'column_12', 'column_13', 'column_14', 'column_15',
'column_16', 'column_17', 'column_18', 'column_19', 'column_20',
'column_21', 'column_22', 'column_23', 'column_24', 'column_25',
'column_26', 'column_27', 'column_28', 'column_29', 'column_30',
'column_31', 'column_32', 'column_33', 'column_34', 'column_35',
'column_36', 'column_37', 'column_38', 'column_39', 'column_40',
'column_41', 'column_42', 'column_43', 'column_44', 'column_45',
'column_46', 'column_47', 'column_48', 'column_49', 'column_50',
'column_51', 'column_52', 'column_53', 'column_54'});

predictorNames = {'column_50', 'column_51', 'column_53',
'column_54'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_52;
isCategoricalPredictor = [false, false, false, false];

% Perform cross-validation
KFolds = 5;
cvp = cvpartition(size(response, 1), 'KFold', KFolds);
% Initialize the predictions to the proper sizes
validationPredictions = response;
for fold = 1:KFolds
    trainingPredictors = predictors(cvp.training(fold), :);
    trainingResponse = response(cvp.training(fold), :);
    foldIsCategoricalPredictor = isCategoricalPredictor;

    % Train a regression model
    % This code specifies all the model options and trains the
model.
    responseScale = iqr(trainingResponse);
    if ~isfinite(responseScale) || responseScale == 0.0
        responseScale = 1.0;
    end
    boxConstraint = responseScale/1.349;
    epsilon = responseScale/13.49;
    regressionSVM = fitrsvm(...
        trainingPredictors, ...
        trainingResponse, ...
        'KernelFunction', 'linear', ...
        'PolynomialOrder', [], ...
        'KernelScale', 'auto', ...
        'BoxConstraint', boxConstraint, ...
        'Epsilon', epsilon, ...
        'Standardize', true);

    % Create the result struct with predict function
    svmPredictFcn = @(x) predict(regressionSVM, x);
    validationPredictFcn = @(x) svmPredictFcn(x);
    % Add additional fields to the result struct

    % Compute validation predictions
    validationPredictors = predictors(cvp.test(fold), :);
    foldPredictions = validationPredictFcn(validationPredictors);

    % Store predictions in the original order
    validationPredictions(cvp.test(fold), :) = foldPredictions;
end

% Compute validation RMSE
isNotMissing = ~isnan(validationPredictions) & ~isnan(response);
validationRMSE = sqrt(nansum(( validationPredictions - response
).^2) / numel(response(isNotMissing) ));
```

B.1 SVMR First Model - plots



*this plot corresponds to SVMR first model



*this plot corresponds to SVMR first model

B.2 SVMR Second Model

```
% SVM REGRESSION second model

%this model calculate predictions for location 52, using info from
%every location of the dataset

clear all
close all

load finaldataset.csv
data=finaldataset;
%first column=month
%second column = day of the month
%rest of columns = solar energy in different locations (52)



%normalization of the values from 0 to 1
data_norm=zeros(8760,54);
for i=3:54
data_norm(:,i)=(data(:,i)-min(data(:,i)))/(max(data(:,i))-
min(data(:,i)));
end

data_norm(:,1:2)=data(:,1:2);


% dividing data into training and testing data
% ratio 75/25

training_data=data_norm(1:0.75*length(data_norm),:);

test_data=data_norm(0.75*length(data_norm):length(data_norm),:);


t2=[test_data(:,1:51), test_data(:,53:54)];

%train the model
[trainedModel, validationRMSE] =
trainRegressionModel_SVMR2(training_data)

%make predictions with test data
yfit= trainedModel.predictFcn(t2);


%creating plots to compare predicted data vs real data
figure
plot(test_data(1:100,50))  %100 hours of real data of column 52
title('SVMR Model')

hold on

plot(yfit(1:100)) %100 hours of predicted data in column 52
xlabel('first 100 hours ')
ylabel('Solar Energy')
legend('real data','predicted data')

%plot 1 month
```

```
figure
plot(test_data(1:720,50))
title('SVMR Model')

hold on

plot(yfit(1:720))
xlabel('1 month ')
ylabel('Solar Energy')
legend('real data','predicted data')

%calculate Root Mean Squared Error between predicted and real data
err_SVMR=immse(test_data(:,50),yfit)
```

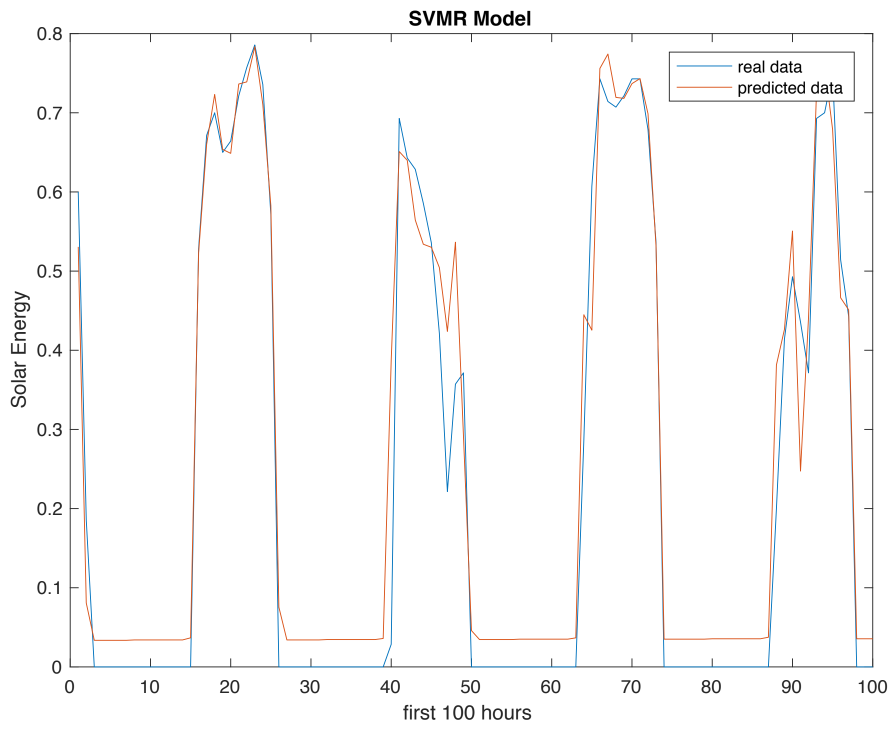
B.2 SVMR Second Model - function training model

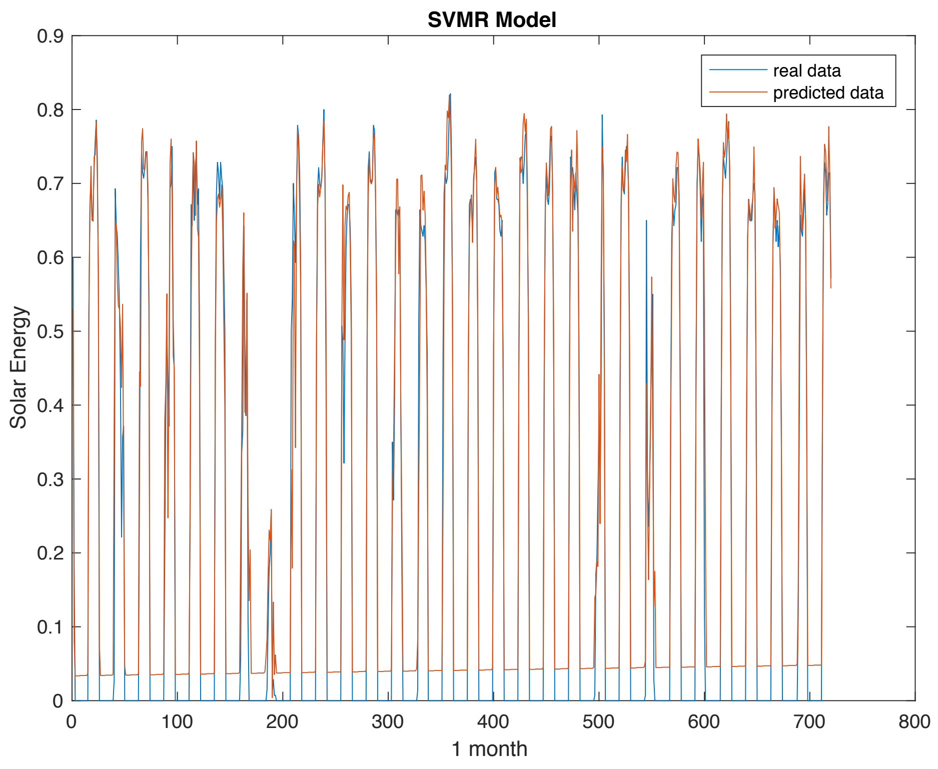
```
function [trainedModel, validationRMSE] =
trainRegressionModel_SVMR2(trainingData)

inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9', 'column_10',
'column_11', 'column_12', 'column_13', 'column_14', 'column_15',
'column_16', 'column_17', 'column_18', 'column_19', 'column_20',
'column_21', 'column_22', 'column_23', 'column_24', 'column_25',
'column_26', 'column_27', 'column_28', 'column_29', 'column_30',
'column_31', 'column_32', 'column_33', 'column_34', 'column_35',
'column_36', 'column_37', 'column_38', 'column_39', 'column_40',
'column_41', 'column_42', 'column_43', 'column_44', 'column_45',
'column_46', 'column_47', 'column_48', 'column_49', 'column_50',
'column_51', 'column_52', 'column_53', 'column_54'});

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4',
'column_5', 'column_6', 'column_7', 'column_8', 'column_9',
'column_10', 'column_11', 'column_12', 'column_13', 'column_14',
'column_15', 'column_16', 'column_17', 'column_18', 'column_19',
'column_20', 'column_21', 'column_22', 'column_23', 'column_24',
'column_25', 'column_26', 'column_27', 'column_28', 'column_29',
'column_30', 'column_31', 'column_32', 'column_33', 'column_34',
'column_35', 'column_36', 'column_37', 'column_38', 'column_39',
'column_40', 'column_41', 'column_42', 'column_43', 'column_44',
'column_45', 'column_46', 'column_47', 'column_48', 'column_49',
'column_50', 'column_51', 'column_53', 'column_54'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_52;
isCategoricalPredictor = [false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false];

% Train a regression model
% This code specifies all the model options and trains the model.
responseScale = iqr(response);
if ~isfinite(responseScale) || responseScale == 0.0
    responseScale = 1.0;
end
boxConstraint = responseScale/1.349;
epsilon = responseScale/13.49;
regressionSVM = fitrsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'linear', ...
    'PolynomialOrder', [], ...
    'KernelScale', 'auto', ...
    'BoxConstraint', boxConstraint, ...
    'Epsilon', epsilon, ...
    'Standardize', true);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames',
predictorNames);
```

```
svmPredictFcn = @(x) predict(regressionSVM, x);
trainedModel.predictFcn = @(x)
svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedModel.RegressionSVM = regressionSVM;
trainedModel.About = 'This struct is a trained model exported from
Regression Learner R2017b.';
trainedModel.HowToPredict = sprintf('To make predictions on a new
predictor column matrix, X, use: \n  yfit = c.predictFcn(X)
\nreplacing ''c'' with the name of the variable that is this
struct, e.g. ''trainedModel''. \n \nX must contain exactly 53
columns because this model was trained using 53 predictors. \nX
must contain only predictor columns in exactly the same order and
format as your training \ndata. Do not include the response column
or any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appregression_exportmodeltoworkspace'')">How to predict using an
exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training
the
% model.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9', 'column_10',
'column_11', 'column_12', 'column_13', 'column_14', 'column_15',
'column_16', 'column_17', 'column_18', 'column_19', 'column_20',
'column_21', 'column_22', 'column_23', 'column_24', 'column_25',
'column_26', 'column_27', 'column_28', 'column_29', 'column_30',
'column_31', 'column_32', 'column_33', 'column_34', 'column_35',
'column_36', 'column_37', 'column_38', 'column_39', 'column_40',
'column_41', 'column_42', 'column_43', 'column_44', 'column_45',
'column_46', 'column_47', 'column_48', 'column_49', 'column_50',
'column_51', 'column_52', 'column_53', 'column_54'});

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4',
'column_5', 'column_6', 'column_7', 'column_8', 'column_9',
'column_10', 'column_11', 'column_12', 'column_13', 'column_14',
'column_15', 'column_16', 'column_17', 'column_18', 'column_19',
'column_20', 'column_21', 'column_22', 'column_23', 'column_24',
'column_25', 'column_26', 'column_27', 'column_28', 'column_29',
'column_30', 'column_31', 'column_32', 'column_33', 'column_34',
'column_35', 'column_36', 'column_37', 'column_38', 'column_39',
'column_40', 'column_41', 'column_42', 'column_43', 'column_44',
'column_45', 'column_46', 'column_47', 'column_48', 'column_49',
'column_50', 'column_51', 'column_53', 'column_54'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_52;
isCategoricalPredictor = [false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false, false];

% Perform cross-validation
KFolds = 5;
```

```
cvp = cvpartition(size(response, 1), 'KFold', KFolds);
% Initialize the predictions to the proper sizes
validationPredictions = response;
for fold = 1:KFolds
    trainingPredictors = predictors(cvp.training(fold), :);
    trainingResponse = response(cvp.training(fold), :);
    foldIsCategoricalPredictor = isCategoricalPredictor;

    % Train a regression model
    % This code specifies all the model options and trains the
model.
    responseScale = iqr(trainingResponse);
    if ~isfinite(responseScale) || responseScale == 0.0
        responseScale = 1.0;
    end
    boxConstraint = responseScale/1.349;
    epsilon = responseScale/13.49;
    regressionSVM = fitrsvm(...
        trainingPredictors, ...
        trainingResponse, ...
        'KernelFunction', 'linear', ...
        'PolynomialOrder', [], ...
        'KernelScale', 'auto', ...
        'BoxConstraint', boxConstraint, ...
        'Epsilon', epsilon, ...
        'Standardize', true);

    % Create the result struct with predict function
    svmPredictFcn = @(x) predict(regressionSVM, x);
    validationPredictFcn = @(x) svmPredictFcn(x);

    % Add additional fields to the result struct

    % Compute validation predictions
    validationPredictors = predictors(cvp.test(fold), :);
    foldPredictions = validationPredictFcn(validationPredictors);

    % Store predictions in the original order
    validationPredictions(cvp.test(fold), :) = foldPredictions;
end

% Compute validation RMSE
isNotMissing = ~isnan(validationPredictions) & ~isnan(response);
validationRMSE = sqrt(nansum(( validationPredictions - response
).^2) / numel(response(isNotMissing) ));
```

B.2 SVMR Second Model - plots





Appendix C

Here is the MATLAB code and functions used in the third tested algorithm: Artificial Neural Networks. It is included only the first example of each model, with a delay of d=2 and algorithm Bayesian Regularization, since the other examples are exactly the same, changing the algorithm and the number of delays.

C.1 ANN First Model

```
%ANN first model
clear all
close all

%making predictions in column 52, using info from every location.
% (NARX)
%d=2 time steps
%algorithm BR

load finaldataset.csv
data=finaldataset;
%first column=month
%second column = day of the month
%rest of columns = solar energy in different locations (52)



%normalization of the values from 0 to 1
data_norm=zeros(8760,54);
for i=3:54
data_norm(:,i)=(data(:,i)-min(data(:,i)))/(max(data(:,i))-
min(data(:,i)));
end

data_norm(:,1:2)=data(:,1:2);


% dividing data into training and testing data
% ratio 75/25

training_data=data_norm(1:0.75*length(data_norm),:);

test_data=data_norm(0.75*length(data_norm):length(data_norm),:);

x_data_input=[training_data(:,1:51),training_data(:,53:end)];
%xdata input is all the columns except the one we want to predict
y_data_target=training_data(:,52);
%ydata target is the column we want to predict


%%%% ANN CODE %%%%

% Solve an Autoregression Problem with External Input with a NARX
Neural Network
% Script generated by Neural Time Series app
% Created 17-May-2018 11:54:54
%
% This script assumes these variables are defined:
%
%   x_data_input - input time series.
%   y_data_target - feedback time series.

X = tonndata(x_data_input,false,false);
T = tonndata(y_data_target,false,false);

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging
problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainbr';  % Levenberg-Marquardt backpropagation.

% Create a Nonlinear Autoregressive Network with External Input
inputDelays = 1:2;
feedbackDelays = 1:2;
hiddenLayerSize = 10;
net =
narxnet(inputDelays,feedbackDelays,hiddenLayerSize,'open',trainFcn
);

% Prepare the Data for Training and Simulation
% The function PREPARETS prepares timeseries data for a particular
network,
% shifting time by the minimum amount to fill input states and
layer
% states. Using PREPARETS allows you to keep your original time
series data
% unchanged, while easily customizing it for networks with
differing
% numbers of delays, with open loop or closed loop feedback modes.
[x,xi,ai,t] = preparets(net,X,{},T);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t,xi,ai);

% Test the Network
y = net(x,xi,ai);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotresponse(t,y)
```

```
%figure, ploterrcorr(e)
%figure, plotinerrcorr(x,e)

% Closed Loop Network
% Use this network to do multi-step prediction.
% The function CLOSELOOP replaces the feedback input with a direct
% connection from the outout layer.
netc = closeloop(net);
netc.name = [net.name ' - Closed Loop'];
view(netc)
[xc,xic,aic,tc] = preparets(netc,X,{},T);
yc = netc(xc,xic,aic);
closedLoopPerformance = perform(net,tc,yc)

% Step-Ahead Prediction Network
% For some applications it helps to get the prediction a timestep
early.
% The original network returns predicted y(t+1) at the same time
it is
% given y(t+1). For some applications such as decision making, it
would
% help to have predicted y(t+1) once y(t) is available, but before
the
% actual y(t+1) occurs. The network can be made to return its
output a
% timestep early by removing one delay so that its minimal tap
delay is now
% 0 instead of 1. The new network returns the same outputs as the
original
% network, but outputs are shifted left one timestep.
nets = removedelay(net);
nets.name = [net.name ' - Predict One Step Ahead'];
view(nets)
[xs,xis,ais,ts] = preparets(nets,X,{},T);
ys = nets(xs,xis,ais);
stepAheadPerformance = perform(nets,ts,ys)


outputfinalNN= cell2mat(y');
for R=1:length(outputfinalNN)
    if  outputfinalNN(R)<0;
        outputfinalNN(R)=0;
    end
end

outputfinalCL= cell2mat(yc');
for R=1:length(outputfinalCL)
    if  outputfinalCL(R)<0;
        outputfinalCL(R)=0;
    end
end

outputfinalAP= cell2mat(ys');
for R=1:length(outputfinalAP)
    if  outputfinalAP(R)<0;
        outputfinalAP(R)=0;
    end
end

%  comparing plots
```

```
figure
plot(y_data_target(feedbackDelays(end):(200+feedbackDelays(end))))

hold on
plot(outputfinalNN(1:200))
legend ('real data',' predicted data')
title('neural network')

%plot neutal network closed loop

figure
plot(y_data_target(feedbackDelays(end):(200+feedbackDelays(end))))

hold on
plot(outputfinalCL(1:200))
legend ('real data',' predicted data')
title('closed loop neural network')


%plot neural network one step ahead prediction

figure
plot(y_data_target(feedbackDelays(end):(200+feedbackDelays(end))))

hold on
plot(outputfinalAP(1:200))
legend ('real data',' predicted data')
title('neural network one step ahead prediction')

%calculate RMSE

err_NN=immse(y_data_target(feedbackDelays(end):end-
1),outputfinalNN)
err_CL=immse(y_data_target(feedbackDelays(end):end-
1),outputfinalCL)
err_AP=immse(y_data_target(feedbackDelays(end):end),outputfinalAP)
```

C.1 ANN Second Model

```
%ANN second model

%making predictions in location 52, using historical data
%from the same location (NAR)

%d=2
%algorithm BR

clear all
close all

load finaldataset.csv
data=finaldataset;
%first column=month
%second column = day of the month
%rest of columns = solar energy in different locations (52)



%normalization of the values from 0 to 1
data_norm=zeros(8760,54);
for i=3:54
data_norm(:,i)=(data(:,i)-min(data(:,i)))/(max(data(:,i))-
min(data(:,i)));
end

data_norm(:,1:2)=data(:,1:2);


% dividing data into training and testing data
% ratio 75/25

training_data=data_norm(1:0.75*length(data_norm),:);

test_data=data_norm(0.75*length(data_norm):length(data_norm),:);


desired_output=training_data(:,52);



%%%% ANN code%%%%


% Solve an Autoregression Time-Series Problem with a NAR Neural
Network
% Script generated by Neural Time Series app
% Created 17-May-2018 14:52:10
%
% This script assumes this variable is defined:
%
%   desired_output - feedback time series.

T = tonndata(desired_output,false,false);
```

```
% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging
problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainbr';  % Levenberg-Marquardt backpropagation.

% Create a Nonlinear Autoregressive Network
feedbackDelays = 1:2;
hiddenLayerSize = 10;
net = narnet(feedbackDelays,hiddenLayerSize,'open',trainFcn);

% Prepare the Data for Training and Simulation
% The function PREPARETS prepares timeseries data for a particular
network,
% shifting time by the minimum amount to fill input states and
layer
% states. Using PREPARETS allows you to keep your original time
series data
% unchanged, while easily customizing it for networks with
differing
% numbers of delays, with open loop or closed loop feedback modes.
[x,xi,ai,t] = preparets(net,{},{},T);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t,xi,ai);

% Test the Network
y = net(x,xi,ai);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotresponse(t,y)
%figure, ploterrcorr(e)
%figure, plotinerrcorr(x,e)

% Closed Loop Network
% Use this network to do multi-step prediction.
% The function CLOSELOOP replaces the feedback input with a direct
% connection from the outout layer.
netc = closeloop(net);
netc.name = [net.name ' - Closed Loop'];
view(netc)
[xc,xic,aic,tc] = preparets(netc,{},{},T);
yc = netc(xc,xic,aic);
closedLoopPerformance = perform(net,tc,yc)
```

```
% Step-Ahead Prediction Network
% For some applications it helps to get the prediction a timestep
early.
% The original network returns predicted y(t+1) at the same time
it is
% given y(t+1). For some applications such as decision making, it
would
% help to have predicted y(t+1) once y(t) is available, but before
the
% actual y(t+1) occurs. The network can be made to return its
output a
% timestep early by removing one delay so that its minimal tap
delay is now
% 0 instead of 1. The new network returns the same outputs as the
original
% network, but outputs are shifted left one timestep.
nets = removedelay(net);
nets.name = [net.name ' - Predict One Step Ahead'];
view(nets)
[xs,xis,ais,ts] = preparets(nets,{},{},T);
ys = nets(xs,xis,ais);
stepAheadPerformance = perform(nets,ts,ys)

%comparing plots



outputfinalNN= cell2mat(y');
for R=1:length(outputfinalNN)
    if  outputfinalNN(R)<0;
        outputfinalNN(R)=0;
    end
end

outputfinalCL= cell2mat(yc');
for R=1:length(outputfinalCL)
    if  outputfinalCL(R)<0;
        outputfinalCL(R)=0;
    end
end

outputfinalAP= cell2mat(ys');
for R=1:length(outputfinalAP)
    if  outputfinalAP(R)<0;
        outputfinalAP(R)=0;
    end
end

%plot neural network

figure
plot(desired_output(feedbackDelays(end):(200+feedbackDelays(end)))
)

hold on
plot(outputfinalNN(1:200))
legend ('real data',' predicted data')
title('neural network')

%plot neutal network closed loop
```

```
figure
plot(desired_output(feedbackDelays(end):(200+feedbackDelays(end)))
)

hold on
plot(outputfinalCL(1:200))
legend ('real data',' predicted data')
title('closed loop neural network')


%plot neural network one step ahead prediction

figure
plot(desired_output(feedbackDelays(end):(200+feedbackDelays(end)))
)

hold on
plot(outputfinalAP(1:200))
legend ('real data',' predicted data')
title('neural network one step ahead prediction')

%RMSE
err_NN=immse(desired_output(feedbackDelays(end):end-
1),outputfinalNN)
err_CL=immse(desired_output(feedbackDelays(end):end-
1),outputfinalCL)
err_AP=immse(desired_output(feedbackDelays(end):end),outputfinalAP
)
```