UiO : Department of Informatics
University of Oslo

Authors: David Fordham, Oskar Våle, Hui Zhang
https://github.com/oskarva/IN5410-project-2

Assignment 2: Machine Learning for Wind Energy Forecasting

IN5410/IN9410 — Energy Informatics: Group 3

Spring 2024

This report outlines our solutions for Assignment 2 on wind energy forecasting by using different machine learning techniques as part of the Energy Informatics course.

1 Task 1

In this task, we focus on the relationship between wind power generation and wind speed. Based on the training data from 01/01/2012 to 31/10/2013 in the file TrainData.csv, we apply machine learning techniques to find the relationship between wind power generation and wind speed. Here, we only use the weather data forecasting of wind speed at 10m above ground level. The machine learning techniques include: linear regression, k-nearest neighbor (kNN), supported vector regression (SVR), and artificial neural networks (ANN).

For this, we use the Linear Regression, K-Nearest Neighbors Regressor, SVR and MLP Regressor methods from the SKLearn library in Python.

Although hyperparameter tuning was not in scope of this assignment, for the kNN method we chose the k according to the suggestion given in [1], which is

$$m = \sqrt{n}, \tag{1}$$

where n is the number of observations of the training data set, and the nearest integer value of m is used as the best k value. For the SVR, the default parameters already outperformed the other methods, and for the ANN, by trial and error, we selected a NN with 2 hidden layers, each with 100 nodes, while using ReLU as the activation function.

The code can be found in the attachments, and the results of the wind power predictions for November 2013 are plotted in figure 1, for all these methods.

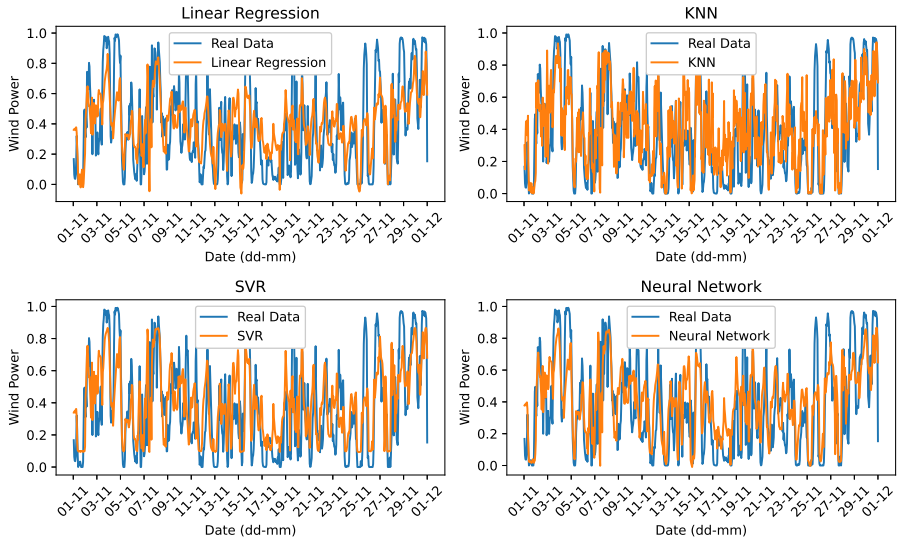

Figure 1: Time series plot with wind power profile for November 2013 and predictions with linear regression, kNN, SVR and ANN models.

Model	RMSE
Linear Regression	0.216384
kNN	0.218835
SVR	0.213783
ANN	0.215743

Table 1: RMSE for linear regression, kNN, SVR and ANN models.

The results seem in general quite good, but it is difficult to appreciate the effectiveness of each method by just looking at the graphical representation of their predictions. We therefore calculate the root mean square error (RMSE) of each method, to quantify their accuracy. The results are shown in table 1.

In general, each model performs in a relatively similar way, with the RMSE remaining the same up to the third decimal point. Going into detail, the best performing method was the SVR with the default parameters of the SKLearn library, and the worst was the kNN, even with the hyperparameter tuning described in equation (1).

These differences arise due to the fundamentally different underlying assumptions and methods present in each model. Linear regressions assume linear relationships between the data; SVR seeks to find a hyperplane that best fits the data points in a continuous space; kNN attempts to approximate the underlying distribution of the data in a non-parametric way; ANN adjust transmission weights between nodes in different layers to reduce learn and reduce the error function, making them highly non-linear.

2 Task 2

Wind power generation may be not only dependent on wind speed, it may be also related to wind direction, temperature, and pressure. In this task, we focus on the relationship between wind power generation and two weather parameters, i.e. wind speed and wind direction.

To do so, we must first calculate the wind direction based on the zonal u_{10} and meridional v_{10} components, which in degrees is

$$\theta = \arctan\left(\frac{v_{10}}{u_{10}}\right)\frac{180^o}{\pi}. \tag{2}$$

We then include θ in the feature list considered for the training process, and proceed to build a Multiple Linear Regression (MLR) model. We then predict the wind power generation for the month of November, and do the same for a simple linear regression considering only wind speed as a feature. The results can be found in figure 2, and the code used has been submitted as an attachment.

Model	RMSE
Linear Regression	0.21638
Multiple Linear Regression	0.21556

Table 2: RMSE for MLR model and simple linear regression.

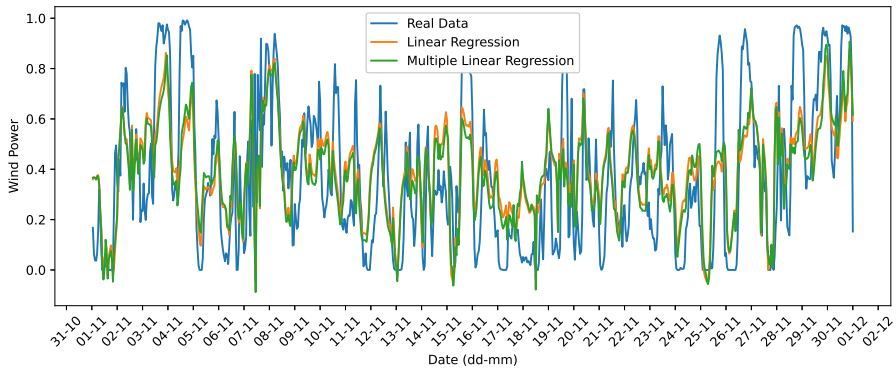

Figure 2: Time series plot with wind power profile for November 2013 and predictions with MLR and simple linear regression.

Additionally, we calculate the RMSE for both models, which can be found in table 2. It is clear, both from the lower RMSE for the MLR model and from a direct observation of the time series curves, that including wind direction improves the wind power forecast, meaning it is a complementary feature to wind speed in this scenario.

3 Task 3

In scenarios where weather data, such as wind speed, is unavailable at the wind farm's location, alternative methods must be employed for predicting wind power production. This part describes our approach to forecasting wind power generation using only the available time-series data on power output. The dataset comprises two columns: TIMESTAMP and POWER. We applied four different machine learning techniques to predict wind power generation for November 2013: Linear Regression (LR), Support Vector Regression (SVR), Artificial Neural Networks (ANN), and Recurrent Neural Networks (RNN).

3.1 Data Preparation for Time-Series Forecasting

For each of these models, we treated the POWER data as a sequential time-series input. To prepare the data for training, we used a lagged window approach where past values of POWER serve as inputs to predict future outputs. The models were trained to predict the power output based on historical values from previous timestamps.

Creating Lagged Features (Input Encoding)

To prepare our dataset for various predictive models, we utilize a method known as **lagging**, which is essential for time-series analysis. This method involves using previous observations to predict future outcomes, with the critical parameter being the **WINDOW_SIZE**.

Definition of WINDOW_SIZE

The **WINDOW_SIZE** parameter defines the number of past observations used to predict the next future point. We use **WINDOW_SIZE** of 1, which means each input vector consists of the immediate past single observation of power output. Mathematically, this can be represented as:

$$X_t = [P_{t-1}] \quad \text{for predicting} \quad P_t \tag{3}$$

where X_t is the input feature vector at time t, and P_{t-1} is the power output at time $t-1$. This approach effectively transforms the original time series into a format suitable for learning, where each data point is dependent on its immediate predecessor.

Output Encoding

Output encoding in time-series forecasting models involves setting the target variable, which the model will learn to predict. In the context of this study, the target for each input vector is the power output at the current timestamp based on the values from the previous timestamps provided by the **WINDOW_SIZE**.

Time Steps and Features

In models like Recurrent Neural Networks (RNN), the input data needs to be specifically formatted to include the notion of **Time steps**:

- **Time steps**: This refers to the number of intervals considered for each input sequence, directly correlated with **WINDOW_SIZE**. Each time step represents a point in the sequence of the data that the model uses to predict the next step.
- **Features per step**: This indicates the number of features considered at each time step. For a simple univariate series like ours, this would typically be one, representing the power output.

The following pseudo-code exemplifies the data reshaping required for an RNN, given our setup:

```
train_X_rnn = train_X.values.reshape(-1, WINDOW_SIZE, 1)
```

This reshaping process prepares the data such that the RNN can process sequences effectively, leveraging its architecture that inherently accounts for the temporal dependencies in the series.

3.2 Model Training

- The **Linear Regression model** was trained to find a linear relationship between past power values and the future power output.
- The **SVR model** utilized a kernel-based method to handle the non-linear patterns in the data.
- The **ANN model** was configured with multiple layers to capture complex relationships in the time-series data.

- The **RNN model**, specifically designed to work with sequences, used its internal state to remember past data points and predict future values.

3.3 Results Visualization and Accuracy Assessment

For visual analysis, we plotted two comprehensive time-series graphs for November 2013, each displaying actual wind power measurements alongside model predictions. The first graph illustrates the forecasts made by LR and SVR models, while the second graph compares the predictions from ANN and RNN models. These visualizations help in directly comparing how each model's predictions align with the actual energy outputs throughout the month.

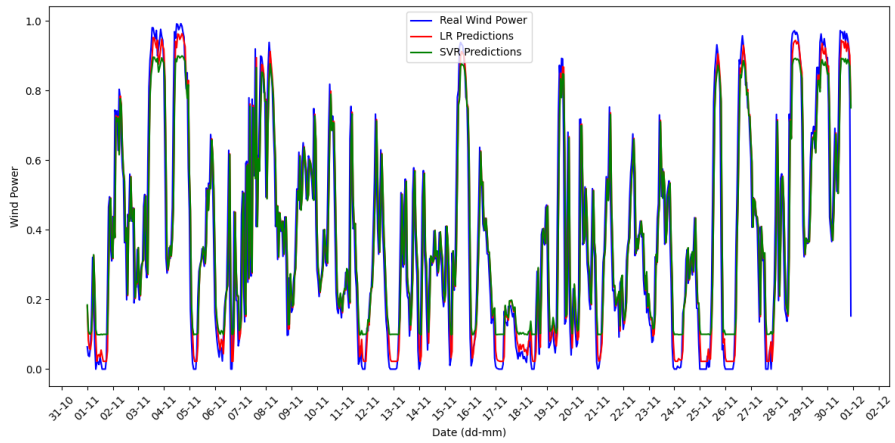

Figure 3: Time series plot with wind power profile for November 2013 and predictions with LR and SVR.

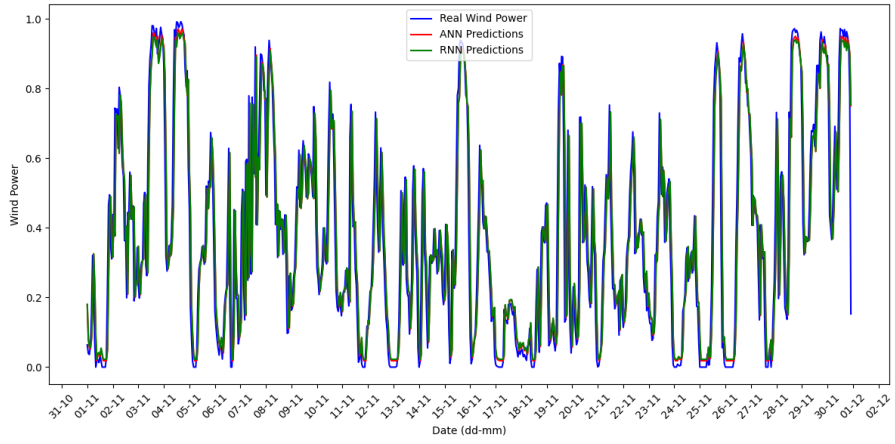

Figure 4: Time series plot with wind power profile for November 2013 and predictions with ANN and RNN.

Additionally, the accuracy of these predictions was quantified using the

Root Mean Squared Error (RMSE), facilitating a direct comparison among the models. The results can be found in table 3.

Model	RMSE
LR	0.125992
SVR	0.129281
ANN	0.125777
RNN	0.125696

Table 3: RMSE for LR, SVR, ANN and RNN models.

The RMSE values are relatively close across all models, indicating that each can predict wind power generation to a similar degree of accuracy. RNN and ANN exhibit slightly better performance, likely due to their ability to effectively model the temporal dynamics in the data. The surprisingly good performance of Linear Regression suggests it is capable of handling some more straightforward forecasting tasks efficiently. The performance of SVR, as well as of the other methods, might improve with better tuning of its parameters and kernel settings. When choosing a model, we'd better consider not only RMSE, but also factors like model complexity, training time, and practical feasibility in real-world applications.

4 Task 4

In this task, we will manually build a neural network with two input nodes, one hidden layer with two nodes, and one output. Its schematics can be found on page 4 of the assignment description.

This neural network uses the sigmoid function as the activation function, which is defined, as well as its derivative, as

$$\sigma(x) = (1 + e^{-x})^{-1}, \tag{4}$$

$$\frac{\mathrm{d}\sigma(x)}{\mathrm{d}x} = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x)). \tag{5}$$

For the error function, we will use the mean square error (MSE), and will apply gradient descent to update all weights during back propagation. This means that, if $x_{n+1} = x_n - \alpha \nabla E(x_n)$, for a learning rate $\alpha \in \mathbb{R}_+$, then $E(x_n) \geq E(x_{n+1})$. Thus, if we update our weights this way, against the gradient of the error function, we guarantee that we progressively improve our results.

To facilitate our calculations, we change the notation slightly to reflect the presence of three layers. We choose the index i for nodes in the input layer, j for the hidden layer and k for the output, with $(i, j, k) \in \mathbb{N}^3$. We also define w_{ij} as the weights involved in the transmissions between the input and hidden layers, and v_{jk} as the ones involved in the hidden to output layer communication. For example, in this notation, weight w_2 would be written as w_{21}, as it involves a communication between input x_2 and hidden node h_1, while weight w_5 would be written as v_{11}.

We begin by randomly selecting initial weight values, ending up with $w_1 = 0.11$, $w_2 = 0.22$, $w_3 = 0.33$, $w_4 = 0.44$, $w_5 = 0.55$, and $w_6 = 0.66$.

We then calculate the outputs of the first forward propagation, which will be

$$h_j = \sigma\left(\Sigma_i w_{ij} x_i\right) = \sigma(z_j) \tag{6}$$
$$\hat{y}_k = \sigma\left(\Sigma_{ij} v_{jk} \sigma(w_{ij} x_i)\right) = \sigma\left(\Sigma_j v_{jk} h_j\right) = \sigma(z_k), \tag{7}$$

which in our case would be

$$h_1 = \sigma(w_1 x_1 + w_2 x_2) \equiv \sigma(w_{11} x_1 + w_{21} x_2) \approx 0.51210 \tag{8}$$
$$h_2 = \sigma(w_3 x_1 + w_4 x_2) \equiv \sigma(w_{12} x_1 + w_{22} x_2) \approx 0.52528 \tag{9}$$
$$\hat{y}_1 = \sigma(w_5 h_1 + w_6 h_2) \equiv \sigma(v_{11} h_1 + v_{21} h_2) \approx 0.65211 \tag{10}$$

We can then calculate the MSE, which is

$$E = \sum_k \frac{(y_k - \hat{y}_k)^2}{2K} = \frac{(y_1 - \hat{y}_1)^2}{2} \approx 0.01157. \tag{11}$$

We must now back propagate, in order to adjust the weights according to the gradient descent technique,

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i}. \tag{12}$$

For this purpose, we will apply the chain rule, to develop a rule for the partial derivatives with respect to the second layer of weights,

$$\begin{aligned} \frac{\partial E}{\partial v_{jk}} = \frac{\partial E}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_k} \frac{\partial z_k}{\partial v_{jk}} &= (y_k - \hat{y}_k)\sigma(z_k)(1 - \sigma(z_k))h_j \\ &= (y_k - \hat{y}_k)\hat{y}_k(1 - \hat{y}_k)h_j, \end{aligned} \tag{13}$$

and to the first layer of weights,

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \sum_k \left[\frac{\partial E}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_k} \frac{\partial z_k}{\partial h_j}\right] \frac{\partial h_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} \\ &= \sum_k \left[(\hat{y}_k - y_k)\sigma(z_k)(1 - \sigma(z_k))v_{jk}\right] \sigma(z_j)(1 - \sigma(z_j))x_i \\ &= \sum_k \left[(\hat{y}_k - y_k)\hat{y}_k(1 - \hat{y}_k)v_{jk}\right] h_j(1 - h_j)x_i. \end{aligned} \tag{14}$$

In the case of our NN, this would explicitly be

$$\frac{\partial E}{\partial w_1} \equiv \frac{\partial E}{\partial w_{11}} = (\hat{y}_1 - y_1)\hat{y}_1(1-\hat{y}_1)w_5 h_1(1-h_1)x_1 \approx 0.0002 \tag{15}$$

$$\frac{\partial E}{\partial w_2} \equiv \frac{\partial E}{\partial w_{21}} = (\hat{y}_1 - y_1)\hat{y}_1(1-\hat{y}_1)w_5 h_1(1-h_1)x_2 \approx 0.0009 \tag{16}$$

$$\frac{\partial E}{\partial w_3} \equiv \frac{\partial E}{\partial w_{12}} = (\hat{y}_1 - y_1)\hat{y}_1(1-\hat{y}_1)w_6 h_2(1-h_2)x_1 \approx 0.0002 \tag{17}$$

$$\frac{\partial E}{\partial w_4} \equiv \frac{\partial E}{\partial w_{22}} = (\hat{y}_1 - y_1)\hat{y}_1(1-\hat{y}_1)w_6 h_2(1-h_2)x_2 \approx 0.0011 \tag{18}$$

$$\frac{\partial E}{\partial w_5} \equiv \frac{\partial E}{\partial v_{11}} = (\hat{y}_1 - y_1)\hat{y}_1(1-\hat{y}_1)h_1 \approx 0.0177 \tag{19}$$

$$\frac{\partial E}{\partial w_6} \equiv \frac{\partial E}{\partial v_{21}} = (\hat{y}_1 - y_1)\hat{y}_1(1-\hat{y}_1)h_2 \approx 0.0181. \tag{20}$$

We can then update the weights as such

$$w_1^* = w_1 - \alpha\frac{\partial E}{\partial w_1} \approx 0.10992 \tag{21}$$

$$w_2^* = w_2 - \alpha\frac{\partial E}{\partial w_2} \approx 0.21962 \tag{22}$$

$$w_3^* = w_3 - \alpha\frac{\partial E}{\partial w_3} \approx 0.32991 \tag{23}$$

$$w_4^* = w_4 - \alpha\frac{\partial E}{\partial w_4} \approx 0.43955 \tag{24}$$

$$w_5^* = w_5 - \alpha\frac{\partial E}{\partial w_5} \approx 0.54293 \tag{25}$$

$$w_6^* = w_6 - \alpha\frac{\partial E}{\partial w_6} \approx 0.65275, \tag{26}$$

and calculate the new output,

$$h_1^* \approx 0.51208 \tag{27}$$

$$h_2^* \approx 0.52525 \tag{28}$$

$$\hat{y}_1^* = 0.65042. \tag{29}$$

Calculating the new MSE, we obtain

$$E^* \approx 0.01131 < E, \tag{30}$$

showing that this first iteration has readjusted the weights in such a way that the error function has decreased, giving us an insight into how a neural network actually works.

We then proceed to program the neural network using the equations derived in this section, for $i \in \{1, 2\}$, $j \in \{1, 2\}$, and $k \in \{1\}$. The stopping condition was ensured through a threshold value between the current and previous MSE. The code can be found attached.

By default, the NN class that we developed generates random weights, but for the purpose of this exercise we consider the same initial weights as those

used in the mathematical section, to see if we obtain the same MSE. The results of the first 10 iterations, with an MSE threshold of $t = 10^{-5}$, can be found in table 4. Note the units of 10^{-3}, used to improve legibility of results in the table format.

Iteration	1	2	3	4	5	6	7	8	9	10
MSE(10^{-3})	11.57	11.31	11.06	10.81	10.57	10.33	10.10	9.87	9.65	9.43

Table 4: MSE for first 10 iterations of developed neural network.

We note that the first two MSEs obtained through the program are exactly the same as those manually calculated, as seen in equations (11) and (30), reassuring us that the NN was implemented congruently with the mathematical equations we have derived. With the NN layout and hyperparameters considered in this exercise, the final MSE is

$$E_f = 0.00038, \tag{31}$$

after adjusting our initial weights.

References

[1] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley, 2012. ISBN: 9781118586006. URL: `https://books.google.no/books?id=Br33IRC3PkQC`.