IN5410
Energy Informatics

ASSIGNMENT 2: MACHINE LEARNING FOR WIND ENERGY FORECASTING

Anna Eliassen
Kristina Othelia Lunde Olsen

Department of Informatics, University of Oslo, Norway
{annaeli, koolsen}@fys.uio.no

May 18, 2020

Abstract

Using machine learning methods for performing wind energy forecasting is an useful tool for today's electricity market and weather forecast. By using historical wind data combined with historical power generation data from wind farms we can train machine learning models to predict the following power potential, making it easier to intelligently manage wind farms and set energy prices, and even predict the coming weather. For the first regression case, where we only consider wind speed and power generation, the SVR model performs better than LR on second, with RMSE=0.214 against RMSE=0.216 respectively. k-NN and FFNN both has RMSE=0.217. In the second case we introduced two wind direction features, resulting in an RMSE=0.209 for MLR, showing that MLR performed better than LR, which has less wind information. Lastly we removed all wind features, only basing the prediction on time-series data, wind power generation data. Now RNN performed best, with RMSE=0.122, followed by FFNN with RMSE=0.123. LR and SVR predicted the new wind power generation a little worse, with RMSE=0.126 and RMSE=0.127, respectively.

Contents

https://github.com/kristinaothelia/IN5410/tree/master/Assignment_2

1 Introduction

Wind energy forecasting is important in many aspects of the electricity market, as well as in weather forecasting. When making decisions in the market, e.g., energy load balance, electricity price, and power grid stability, it's critical to have good/reliable models that can predict the future trends as accurately as possible.

In this assignment we will use wind power history data to train machine learning models to generate wind power forecasts for a given time period. We will look into linear regression, multiple linear regression, k-nearest neighbors, support vector regression, as well as artificial neural networks. All code were done in Python, using Scikit Learn and Keras, including grid search cross validation to tune hyperparameters for our models.

With the provided wind power data we can build models which learns the relationship between weather features and the generated power at the wind farm. Then we use the trained model to forecast wind power generation for a given time period. In some situations, we don't have weather features available for predicting wind power. Therefore, we will also investigate how some of the models perform when the model is trained only using previous power measurements to predict next the power measurements. Since we also have data of the true generated wind power for the same time period we are making predictions for, we can compare the true and the predicted wind power to evaluate our models. This is mainly done with the metric RMSE.

2 Project Data

The data used in this project contains necessary weather data for performing machine learning methods for wind energy forecasting. For a more detailed description of the data files, see Appendix A. The normalized data (to preserve anonymity) is from a wind farm in Australia, and the weather input information is from the European Centre for Medium-range Weather Forecasts (ECMWF - ecmwf.int).

The file **TrainData.csv** consists of six wind features columns and one column with generated wind power. The first three (of six) columns represent the zonal (West-East projection) and meridional (South-North projection) components of the wind forecast, as well as the measured wind speed at 10 meter above ground level. The next three columns are the same three measurements, just at 100 meter above ground level. The dataset provides data for the time period 01.01.2012 to 31.10.2013.

The file **WeatherForecastInput.csv**, consist of the same six wind feature columns as the previous described file. The file provides data for the time period of November 2013, the month after **TrainData.csv** ends. The last set of data, **Solution.csv**, holds the true wind power measurements for the time period of November 2013. After our model is trained using the training data, we can use the model to predict the wind power for November 2013, by using the wind features in **WeatherForecastInput.csv**. We can then check our predictions against the true wind power measurements.

3 Machine Learning

The subject area known as Artificial Intelligence (AI), contains many different techniques that tries to make computer and data programs as intelligent as possible. One of these techniques is called Machine Learning (ML), which is an interdisciplinary field combining physics, mathematics and informatics. Within ML there are many different methods which can be used to make computers and data programs learn and make intelligent decisions based on empirical data. Supervised learning, is when the algorithms is given a set of labeled training data that they can use to relate different input variables to the correct output variable, which is what we are going to use in this project.

Since the power generation we will study in this project is given on a numerical, continuous timescale, we will use the machine learning models suitable for regression analysis. A regression analysis looks at the correlation between a dependent variable (also known as response variable) and one or more independent variables, and can show the degree to which one variable varies with another variable.

3.1 Linear Regression | LR and MLR

Linear regression (LR) is one method which may be used when the response variable is measured on a continuous scale. If we have several explanatory variables, we may use Multiple Linear Regression (MLR), which can be written as equation (1), where y is our response variable. $x_1...x_n$ is the set of n explanatory variables and ϵ represents the error of the model. The regression coefficients, $\beta_1...\beta_n$, provides the relationship between the explanatory variables and the response variable.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \epsilon, \tag{1}$$

For simple LR, where n=1, equation (1) are often written as $y_i = \alpha + \beta x_i + \epsilon_i$, with y-intercept α.

In this project we have implemented the built-in scikit package for Linear Regression, and from the implementation point of view, this is basically Ordinary Least Squares (OLS) [5], and can be used for both simple linear regression and multiple linear regression. When using OLS, the goal is to minimize the sum of the squared errors. Thus, the objective function for OLS (one predictor) may be written as

$$MIN \sum_{i=1}^{n} (y_i - w_i x_i)^2,$$

where y_i is the target, w_i is the coefficient, and x_i is the predictor (explanatory variable).

3.2 k-Nearest Neighbor | k-NN (supervised)

The k-nearest neighbor (kNN) is probably one of the most basic and intuitive supervised machine learning techniques, yet it turns out to be very effective in many cases. It is most commonly used for classification situations, but can also turn out useful in regression problems. The technique uses some known neighboring values, to calculate which value the new data point should have [8].

The first step in the kNN method/algorithm, is to calculate the distance between the new data point, and each of the training data points. The distance can be calculated using either Euclidian, Manhattan or Hamming distance, depending on the dataset. For continuous data the Euclidian or Manhattan distance may be used, while Hamming distance is used for categorical data. In this project we have used Manhattan distance, since the data we studied (described in section 2) is continuous. The Manhattan distance, d, is calculated as

$$d = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2},$$

where x corresponds to the new point, and y is an existing point [8]. After calculating the distance, the value of the new point can be calculated using the k nearest neighbors. To choose the optimal value for k, can be done by using grid search techniques or by comparing how different k-values affect the error using our training set and validation set. Choosing a too low k-value can result in overfitting on the training data, which can lead to a high error on the validation data. Plotting the error in the validation data against a number of k-values, we can determine for which k we get the lowest error.

3.3 Support Vector Regression | SVR

Like kNN, Support Vector Machine (SVM) is another machine learning technique that is well known for classification problems. Less known/used is the technique Support Vector Regression (SVR), which can be useful when solving regression problems [7].

The SVR implementation provides more flexibility than basic OLS, since we can decide/choose an acceptable error (epsilon, ϵ) for our predictions. In addition, we can use another hyperparameter to tune which tolerance we should have for falling outside of the acceptable error range. In contrast to OLS, the SVR technique tries to minimize the l2-norm of the coefficient vector. An SVR objective function may be expressed as

$$MIN\frac{1}{2}||w||^2 + C\sum_{i=1}^{n}|\xi_i|,$$

with constraints described as

$$|y_i - w_i x_i| \leq \epsilon + |\xi_i|.$$

We notice that the error term now is handled in the constraints, and by tuning the maximum acceptable error, ϵ, we can influence the desired accuracy for our SVR model [7]. To further "optimize/improve" the SVR model, we can introduce slack variables to account for the data points that fall outside the error margin, described by ξ. This gives us a new hyperparameter (C), which can be used to tune the tolerance for points falling outside of ϵ. An increase of C results in a higher tolerance for points outside the error margin.

A basic illustration of SVR is given by figure 1, where the bold line represents what is known as the hyperplane. The two dotted lines is called decision boundaries, and is decided by the size of ϵ.

To solve the problem discussed in this project, we have used scikit-learns built-in function for SVR. For a more thoroughly description of how to implement this function, the documentation is listed in reference [6].

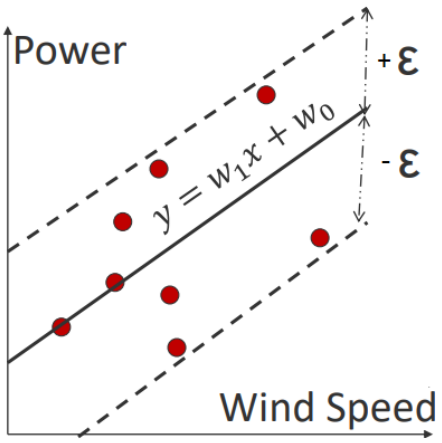
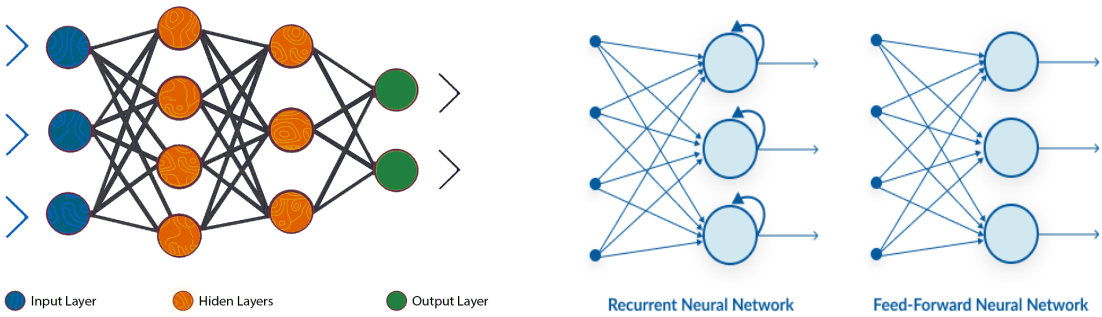


Figure 1: Illustrative Example of SVR (linear) with Slack Variables

3.4 Artificial Neural Networks (supervised)

Neural Networks (NN) are supervised learning techniques that are inspired by the design and function of biological neural networks. By using calculations structured as interconnected groups of artificial neurons, the NN techniques emulate the biological neural networks [3]. These artificial neurons may also be referred to as nodes or units. Figure 2 provides a basic representation of the two NN we will look further into; Feedforward neural networks (FFNN) and Recurrent neural networks (RNN).



(a) Flow of FFNN

(b) RNN (left) vs. FFNN (right)

Figure 2: Simple illustration of FFNN and RNN [3].

Feed Forward Neural Networks | FFNN

Feed Forward Neural Networks (FFNN), is the most "basic" and "simple" implementation of a neural network. As illustrated in figure 2a, the FFNN implementation consists of three different layers; the input layer (blue), one or more hidden layers (orange) and the output layer (green). In FFNN architecture, there are no inter-layer connections or backwards connections. All the nodes in layer i connects to every node in layer i+1, which means the neural network is fully connected [4]. Advantages of a FFNNs is that they use activation functions to introduce non-linear properties to the network, and they are capable of learning any non-linear function. All inputs and outputs are independent of each other, and FFNNs have the capacity to learn weights that map any input to the output.

Recurrent Neural Network | RNN

As we can see in figure 2b, RNN has a recurrent connection on the hidden layer. By adding this looping constraint, we ensure that sequential information is captured in the input data [3]. As a result, we can say that the RNN architecture has a kind of memory, which means it can remember the history information. Some variants of RNN can have a pretty long memory, which may be useful. Other advantages of RNN, is that we can share parameters across different time steps. This is called parameter sharing, and results in fewer parameters to train, which decreases the computational cost. A common problem in all the types of neural networks, is the vanishing and exploding gradient problem. Deep RNNs also suffers from this issue, and should be taken into consideration when implementing deep RNNs. Working with time series data, neural networks are in many cases considered to perform bad. However, RNNs is usually pretty good at predicting sequences or time series data.

3.5 Tuning hyper-parameters of an estimator | GridSearch Cross-Validation

Most machine learning models have hyper-parameters, which are parameters that are not determined by the model itself, like the weights and biases of a NN. These parameters have to be given values beforehand, and are often hard to determine. To find the optimal hyper-parameters for our machine learning algorithms, we chose to use scikit-learn's built-in method **GridSearchCV** [2].
This method search the user-determined hyper-parameter space, and finds the optimal cross validation score. A certain set of hyper-parameters will often have a larger impact on the models computational performance and predicted outcome. Others are less important, and it's sufficient to use default values. Any parameter needed when constructing a model, can be optimized using GridSearchCV.

3.6 Model evaluation | RMSE & R^2

When performing regression analysis we can evaluate the performance of the model by considering the root mean squared error (RMSE). The RMSE is the square root of the mean squared error (MSE). The MSE is a measurement of how close a fitted line is to the data points, or the error between them.

$$\text{RMSE}(\boldsymbol{z}, \tilde{\boldsymbol{z}}) = \sqrt{MSE(\boldsymbol{z}, \tilde{\boldsymbol{z}})} = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2} \tag{2}$$

Where $\boldsymbol{z}$ is the data and $\tilde{\boldsymbol{z}}$ is the model (fitted data). With other words, $\tilde{z}_i$ is the predicted value of the i-th sample and z_i is the corresponding true value. The RMSE gives an average value for the distance between an known data point and the predicted data point. The closer the result is to zero, the better the model. In this project we used scikit-learn's built-in method **mean_squared_error** [1], before taking the square root of the result.

The statistical coefficient of determination, R^2, score is used for indicating how well the model is predicting the actual values in the data set. The closer the R^2 score is to 1, the better the model. The R^2 score function is given by eq. (3), where the mean value of $\boldsymbol{z}$ is defined in eq. (4).

$$\text{R}^2(\boldsymbol{z}, \tilde{\boldsymbol{z}}) = 1 - \frac{\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2}{\sum_{i=0}^{n-1}(z_i - \bar{z}_i)^2} \tag{3}$$

$$\bar{z} = \frac{1}{n}\sum_{i=0}^{n-1} z_i \tag{4}$$

4 Wind energy forecasting

For task 1 and 2, we only used wind data at 10 meter above ground level.

4.1 Task 1 | Wind power generation and wind speed

In the first task, we used four machine learning (ML) methods, LR, k-NN, SVR and FFNN to make wind energy forecasting based on the relationship between wind power generation and wind speed, at 10 meters above ground level. We therefor only used the POWER and WS10 columns from **TrainData.csv**, **WeatherForecastInput.csv** and **Solution.csv**.

We started by making models for all ML methods, using the training data, where the wind power were the target, WS10 were the feature, and the model prediction were made from the weather forecast input (WS10). When all predictions were made, we compared the result with the true wind power measurements, before making .csv datafiles for all predictions. We also calculated R^2 and RMSE metric to evaluate and compare the prediction accuracy among the ML methods, as well as comparison plots, as seen in the Result section.

4.2 Task 2 | Multiple Linear regression

In the second task, we expanded the wind features to include the zonal (U10) and meridional (V10) component of the wind forecast. This is because the wind power generation may not only be dependent on wind speed, it may also be 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).

The predicted wind power production were saved in the file **ForecastTemplate2.csv**. Lastly, we compared the prediction accuracy using MLR with the LR results from Task 1, where the prediction only were based on wind speed. This were done by calculating RMSE and plotting the MLR and LR predictions against the true wind power measurements.

4.3 Task 3 | Regression Without Predictors

In the third and final task, we wanted to study power forecasting without any of the weather features. We removed all the weather features from the training data, leaving just the power column.

With only one column left, we needed to split up the training data in such a way that we still have a set input data and a corresponding output. This was done using a function, that returned X_train and y_train, corresponding to the input data and the output data respectively. The columns (”number of features”) of the input data depends on how many previous time steps we would want to use to predict the value at time, t. If we had a training set, data = [1, 2, 3, 4, 5, 6], and wanted two previous steps to predict the outcome, the function would then split the data set as illustrated in table 1.

Table 1: Splitting of training data to create input and output variables

X_{1t-2}	X_{t-1}	Y_t
1	2	3
2	3	4
3	4	5
4	5	6

The different regression algorithms can then use this training data to learn the relationship between the previous power measurements and output. The methods used are using different algorithms for this, but we provided them the same training data using 10 ”look back” features. Then we used the model to make predictions on the test data.

5 Results & Discussion

In this section the main results of the three different cases we investigated of the timeseries data is presented. Information about how to run the Python code, and hyper parameters for each method, can be found in Appendix B.

5.1 Simple Regression | Power forecasting using wind speed

In table 2 we have the calculated RMSE and R^2 scores for the models. We can see that the results only have small deviations from each other, indicating that they performed the predictions quite similar. All models have an RMSE $\approx$ 0.21 and $R^2 \approx 0.4$, meaning that the wind power generation predictions could have been better. The SVR model gives the lowest RMSE score, which indicates that SVR performed best of the four models tested in task 1, followed by Linear Regression. However, it's important to remember that SVR have an tendency to over-fitting the data. Surprisingly the Neural Network gave the worst prediction, but this could be caused by bad tuning of hyper parameters. The reason why LR did so well could be caused by a fairly linear relationship between wind speed and generated wind power.

The main variation in the result comes from the models themselves, as their algorithm have different ways of finding the relationship between the wind power generation and the wind speed

Table 2: Comparing the prediction accuracy based on RMSE and R^2, for task 1.

Method	RMSE	R^2
Linear Regression	0.216	0.454
k-Nearest Neighbors	0.217	0.452
Support Vector Regression	0.214	0.464
Feed Forward Neural Network	0.217	0.454

5.1.1 Linear Regression | LR

From figure 3 we can see that the predicted wind power for November 2013 is a bad match. The overall shape is similar to the real generated wind power, but the linear regression model are not able to predict the detailed oscillation, extreme points, for the generated wind power.

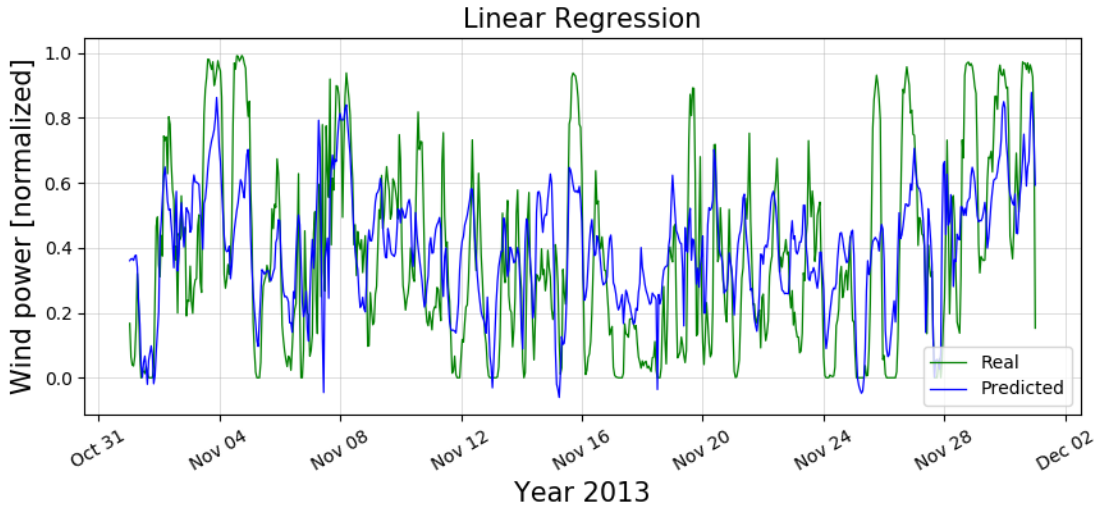


Figure 3: Wind power prediction based on the LR model

5.1.2 K-Nearest Neighbors | kNN

From figure 4 we can see that the predicted wind power for November 2013 is a bad match. The overall shape is similar to the real generated wind power, but the k-NN model are not able to predict the detailed oscillation for the generated wind power, although we can see that the result are a little better than with the LR model. We used k=150, when using k-NN it's important to remember that a high k-value can result in over fitting, but this varies from data set to data set. Performing a cross-validation check is recommended, one can be found in our source code.

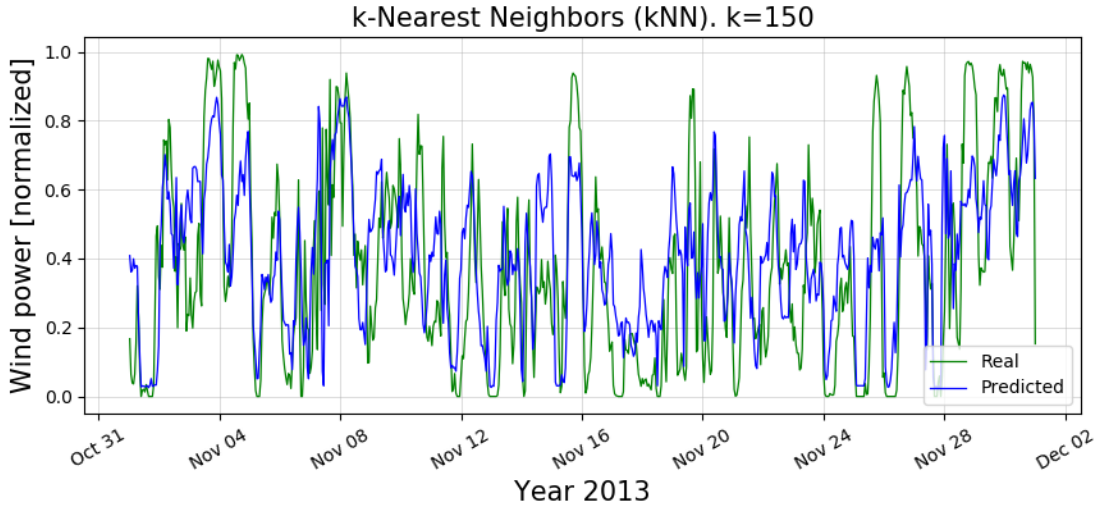


Figure 4: Wind power prediction based on the k-NN model

5.1.3 Support Vector Regression | SVR

The SVR models scored the best accuracy metrics, both in fitting the data and predicting. We can see from figure 5 that the SVR model are able to predict the oscillations, extreme points, a little better than the other models. Compared to for instance Simple Linear Regression, this is quite expected because tuning of the hyperparameters for penalty cost and maximum error optimizes the function to making the best prediction. In addition, we see that the 'rbf' kernal was preferred over a linear kernel.

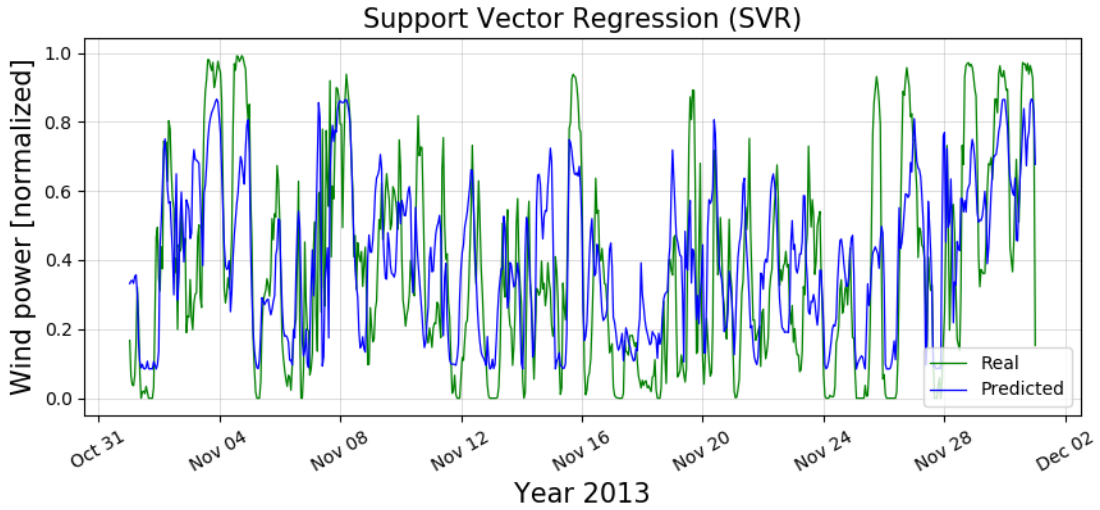


Figure 5: Wind power prediction based on the SVR model

5.1.4 Feed Forward Neural Network | FFNN

The FFNN model shares the worst score with k-NN model. The prediction seen in figure 6 shows a similar prediction tendency as the other models, where the main issue lies in predicting the wind power oscillations, the extreme points. The hyper parameters used are listed under best parameters in Appendix B, but a better set of parameters to test could give a better end result.

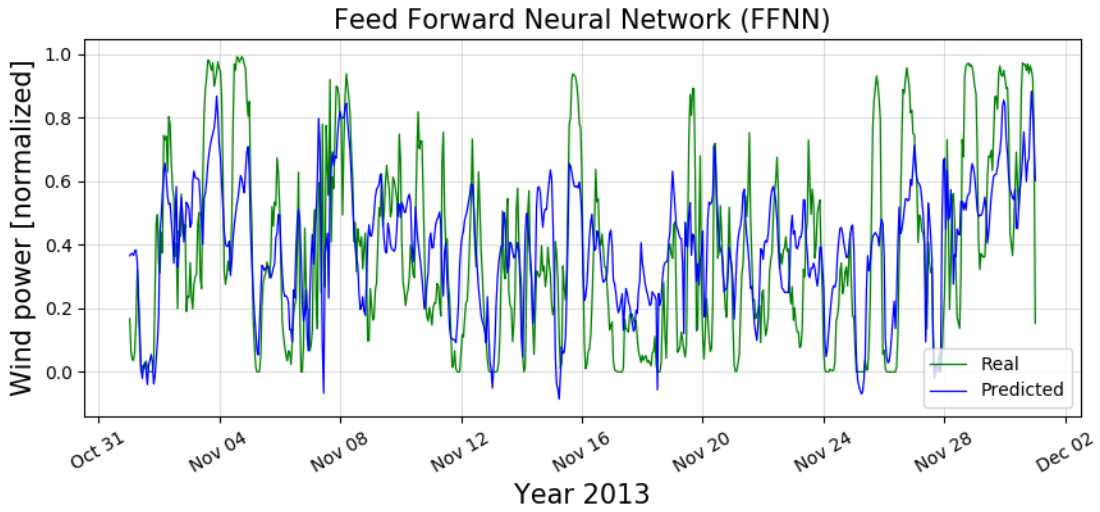


Figure 6: Wind power prediction based on the FFNN model

5.2 Multiple Linear Regression
Wind power forecasting using wind speed and wind direction

Table 3 and figure 7 shows the predicted wind power for November 2013, with the use of Linear Regression (LR) and Multiple Linear Regression (MLR). The figure also shows the true generated wind power for this time period. We can see from both the table and the figure that MLR performs better than LR, mainly in the way MLR predicts extreme points. The difference between the two linear regression models are that LR only makes predictions based on wind speed, while MLR includes two wind directions features. By including the wind direction in the model, the prediction accuracy were increased (lower MSE, higher R^2), so we can conclude that wind direction plays an role in wind power forecasting. We only include wind speed and direction at 10 meter above ground, it's makes sense to assume that including the wind speed and direction at 100 meter above ground could have an bigger impact on the model accuracy. Assuming that the wind farm consists of windmill, they are normally quite tall, so the main source of wind impact making the windmills generate power does not occur at 10 meters above ground. At higher altitudes, the wind speed are also more stable.

Table 3: Prediction accuracy based on RMSE and R^2, for task 2.

Method	RMSE	R^2
Linear Regression	0.216	0.454
Multiple Linear Regression	0.209	0.493

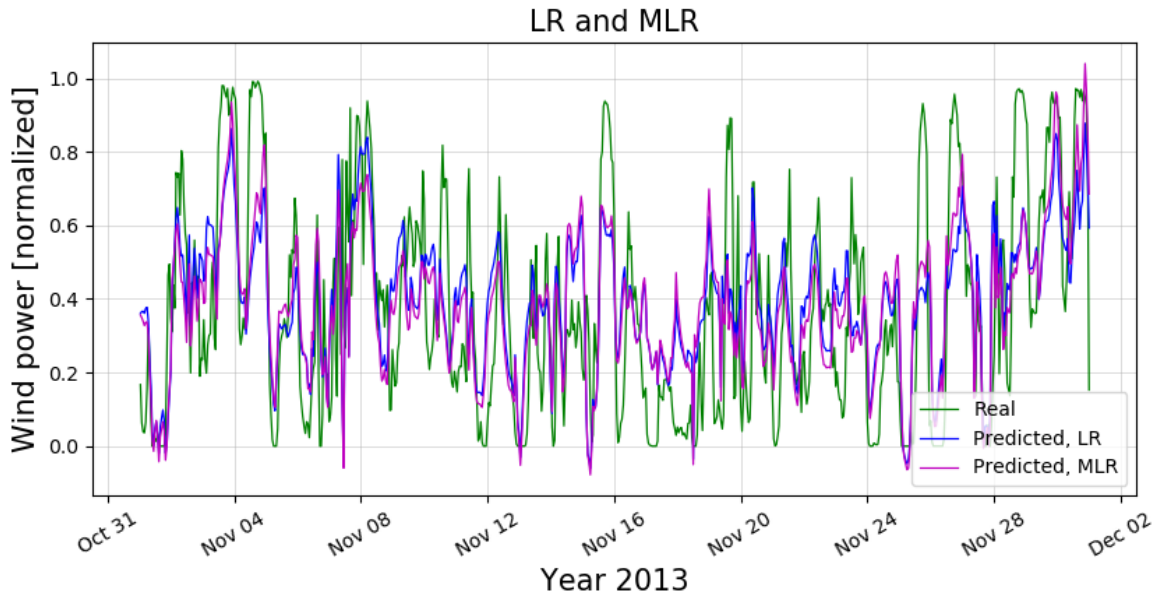


Figure 7: Wind power prediction based on the LR and MLR models. Where MLR includes two more wind features (U10 and V10), LR are only dependent on wind speed (WS10).

5.3 Regression without features. LR, SVR, FFNN and RNN Wind power forecasting without weather data

The result from our regression analysis without weather features, turned out a lot better than expected. Initially we thought the RMSE would be worse compared to when we used wind speed in Task 1. As seen in table 4, as well as figure 8 and 9, the models now predicted the generated wind power for November 2013 very well. The cause has to be because of the way training and testing data are generated, as well as only using time-series data. The model are now making new predictions based on the former generated wind power alone, up to the time of interest.

Table 4: Comparing the prediction accuracy based on RMSE and R^2, for task 3.

Method	**RMSE**	**R^2**
Linear Regression	0.126	0.815
Support Vector Regression	0.127	0.813
Feed Forward Neural Network	0.123	0.823
Recurrent Neural Network	0.122	0.827

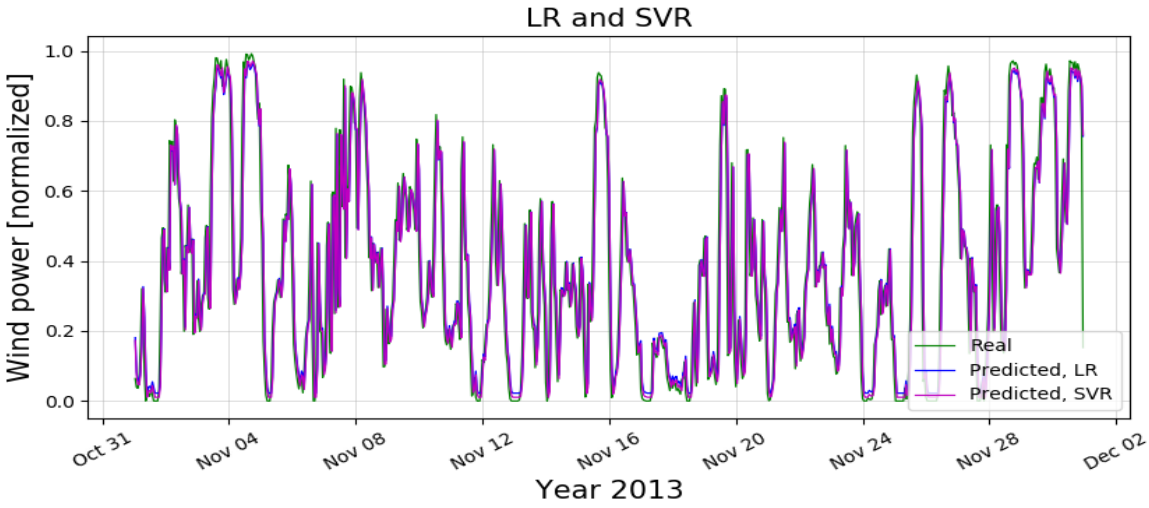


Figure 8: LR and SVR

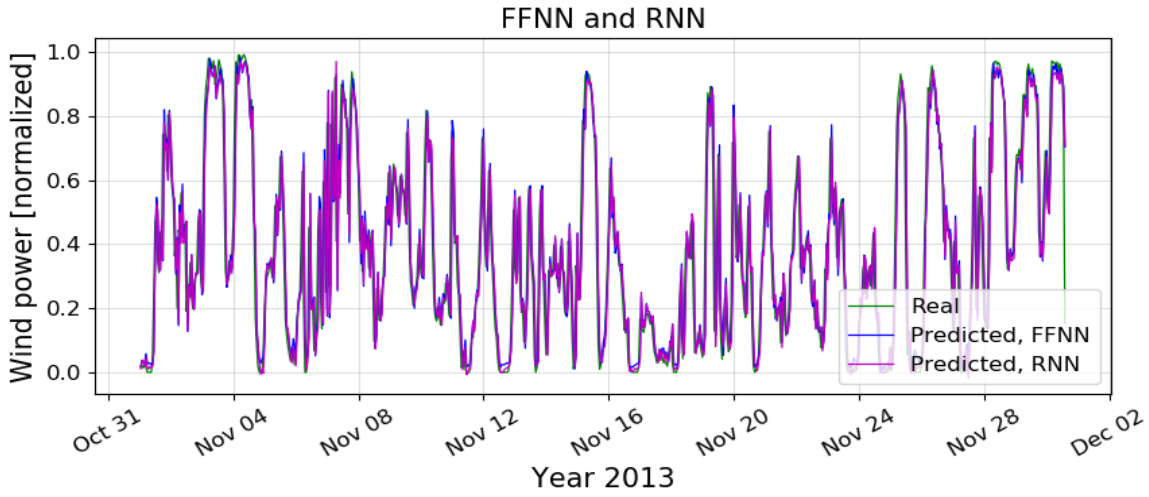


Figure 9: FFNN and RNN

6 Conclusion

In this project we have applied supervised machine learning methods on time series data from a wind farm in Australia. We have investigated how different ML performs when using only one feature (wind speed) and two features (wind speed and direction). We have also looked into the case when no weather features are available, and we had to use previous power data to forecast the power generation. In this last prediction, all models predicted the output very well, with RMSE$\approx$ 0.12.

The models applied to predict power generation with only wind speed as input variable, had almost equally performance, with RMSE$\approx$ 0.21. When also providing the wind direction as input variable, the MLR model had a slightly lower rmse. So in this case, the inclusion of wind direction did not significantly improve the power forecast compared to just wind speed alone, which can be due to the location of the wind farm and/or the placement of the wind turbines related to most frequent wind direction. It can also be a consequence of only looked at features 10m above ground level, since windmills are tall, weather features at higher altitudes could have an larger impact. The equation $P = \frac{1}{2}\rho A V^3$ shows the strong relation between the wind speed and wind power. If more features, such as air density (which depends on T), the wind models would most likely perform better.

References

[1] Scikit Learn. *sklearn.metrics.mean$_s$quared$_e$rror*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html. Visited: 7. May 2020.

[2] Scikit Learn. *sklearn.model$_s$election.GridSearchCV*. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Visited: 7. May 2020.

[3] ARAVIND PAI. *CNN vs. RNN vs. ANN – Analyzing 3 Types of Neural Networks in Deep Learning*. https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/. Published: 17. February 2020, Visited: May 18, 2020.

[4] Adrian Rosebrock. *A simple neural network with Python and Keras*. https://www.pyimagesearch.com/2016/09/26/a-simple-neural-network-with-python-and-keras/. Published: 26. September 2016, Visited: May 18, 2020.

[5] scikit-learn.org. *Documentation for Linear Regression in python by scikit*. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html. Visited: May 18, 2020.

[6] scikit-learn.org. *Documentation for Support Vector Regression in python by scikit*. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html. Visited: May 18, 2020.

[7] Tom Sharp. *An Introduction to Support Vector Regression (SVR)*. https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2. Published: 3. Mars 2020, Visited: May 18, 2020.

[8] AISHWARYA SINGH. *A Practical Introduction to K-Nearest Neighbors Algorithm for Regression (with Python code)*. https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/. Published: 22. August 2018, Visited: May 18, 2020.

Appendix A | Data description

The files to be used for this assignment have all the necessary input weather forecasts, past wind power measurements, as well as templates for submitting the forecasts, for the assignment. The various files include:

TrainData.csv: This file gives the set of data that can be used to find a relationship between weather forecasts inputs (wind speed forecasts at 10m and 100m above ground level) and observed power generation. These data cover the period from 1.1.2012 to 31.10.2013.
Variables (columns):

- TIMESTAMP: time stamps giving date and time of the hourly wind power measurements in following columns. For instance, "20120708 13:00" is for the 8th of July 2012 at 13:00
- POWER: measured power values (normalized)
- U10: zonal component of the wind forecast (West-East projection) at 10m above ground level
- V10: meridional component of the wind forecast (South-North projection) at 10m above ground level
- WS10: wind speed at 10m above ground level
- U100: zonal component of the wind forecast (West-East projection) at 100m above ground level
- V100: meridional component of the wind forecast (South-North projection) at 100m above ground level
- WS100: wind speed at 100m above ground level

WeatherForecastInput.csv: This file gives the set of input weather that can be used as input to predict wind power generation for the whole month of 11.2013. These include wind speed forecasts at 10m and 100m above ground level. These data cover the period from 1.11.2013 to 30.11.2013.
Variables (columns):

- TIMESTAMP: time stamps for the wind forecasts
- U10: zonal component of the wind forecast (West-East projection) at 10m above ground level
- V10: meridional component of the wind forecast (South-North projection) at 10m above ground level
- WS10: wind speed at 10m above ground level
- U100: zonal component of the wind forecast (West-East projection) at 100m above ground level
- V100: meridional component of the wind forecast (South-North projection) at 100m above ground level
- WS100: wind speed at 100m above ground level

Solution.csv: This file gives the true wind power measurements for the whole month of 11.2013, which will be used to calculate the error between your forecasts and the true measured wind power.
Variables (columns):

- TIMESTAMP: time stamps for the wind power measurements, corresponding to the forecasts to be compiled in ForecastTemplate.csv
- POWER: true wind power measurement (normalized)

ForecastTemplate.csv: This file gives the template for submitting your forecasts for the whole month of 11.2013. For each question, this file should be accordingly renamed.
Variables (columns):

- TIMESTAMP: time stamps for the wind power forecast values to be generated
- FORECAST: your forecast values

Appendix B | Model parameters & How to run the python code

——— Model parameters ———

In table 5 we have listed all parameters used in sklearn.model_selection.GridSearchCV() for k-NN, SVR and FFNN in Task 1. In table 6 we have listed all tested parameters for RNN for Task 3, as well as the results for SVR and FFNN. We would recommend to test more parameters for RNN.

Table 5: GridSearchCV for k-Nearest Neighbors, Support Vector Regression and FF Neural Network.

Parameter	Tested values	Best fit
k-Nearest Neighbors		
n_neighbors	2, 5, 10, 50, 100, 200, 300, 400, 500, 600, 850, 1000	600
weights	'uniform', 'distance'	'uniform'
p	1, 2	1
Support Vector Regression		
C	0.001, 0.01, 0.1, 1.0	0.01
kernel	'rbf', 'linear', 'sigmoid'	'rbf'
gamma, γ	'scale', 'auto'	'scale
epsilon, ε	0.01, 0.1, 1.0	0.1
Feedforward Neural Network		
activation	'relu', 'logistic'	'relu'
solver	'sgd', 'adam'	'adam'
alpha, α	0.0001, 0.001, 0.01, 0.1	0.01
learning_rate	'constant', 'adaptive'	'adaptive'
learning_rate_init, η	0.0001, 0.001, 0.01, 0.1	0.0001
max_iter	500, 1000, 1500	1500

Table 6: GridSearchCV for task 3, SVR and FFNN and RNN.

Parameter	Tested values	Best fit
Support Vector Regression	Same as table 5	C: 0.001, kernel: 'linear'
		ϵ: 0.01, γ: 'scale'
Feedforward Neural Network	Same as table 5	activation: 'relu', solver: 'sgd'
		α: 0.01, learning_rate: 'adaptive'
		learning_rate_init: 0.1, max_iter: 500
Recurrent Neural Network		
activation	'relu', 'sigmoid'	'relu'
batch_size	1, 6, 16	6
epochs	5, 10	10
optimizer	'adam', 'sgd'	'adam'
units	3, 15, 30	30

——— How to run the Python code ———

In a terminal, run the code: **main.py -T -M -X -W**

Give one of the arguments for "-T": -1, -2, -3 (Task 1, Task 2, Task 3).
Give one of the arguments for "-M": -L, -K, -S, -F (LR, kNN, SVR, FFNN).
The "-M" argument tells which machine learning method to run, and are only used for **Task 1**.

The optional argument -X will create, save and show plots for the chosen task.
The optional argument -W will print out possible warnings that occurred.

You might need to install: **pip install keras tensorflow**