Machine Learning for Wind Energy Prediction - Possible Improvements over Traditional Methods

Finn Erik Sølverød

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

May 25, 2017

Contents

1 Abstract

This thesis explores various regression methods for predicting wind energy generation in wind parks containing several turbines using historical data, with an emphasis on determining whether Machine Learning algorithms might improve on more traditional methods. As well as how the selection of input data affects the Machine Learning algorithms relative to the traditional methods.

This thesis compares the predictive qualities of several statistical regression methods when applied to the problem of estimating power produced by wind parks. The algorithms explored are Linear Regression, K Nearest neighbours Regression as well as the Machine Learning algorithms Support Vector Regression and a Multi-layer Perceptron used as a regressor. These regression methods were used on various subsets of a data-set containing wind speed and wind power information across a large amount of wind parks. The parks are located across the western United States of America, and have data for the years 2004 to 2006.

The thesis starts by describing the content of the data used in the regressions. The data-set is a large number of wind parks with speed and power generation data in 10 minute increments, for the covered years 2004-2006. Then the thesis lists the tools used in creating the regressions and executing the analysis. Further on there will be an overview of how each of the four algorithms generate their regressions', as well as how to find their relative performance using the error metric Mean Square Error. How this error metrics relates to the standard deviation is also mentioned.

Following this we have a presentation of data in the form of a series of graphs detailing how well the methods (Linear Regression, K Nearest neighbours, Support Vector Regression and Multi-layer Perceptron) perform, in the thesis' three main test situations.

The first test situation is the problem of predicting power generation using measured current wind speed. This is done by creating regressions, and testing the predictive qualities of those regressions.

The second test situation handles the problem of trying to generate a prediction of future power generation in wind parks by creating regressions by using historical power generation data from the exact same wind parks, by looking some number of time-steps into the past. This method is also known as time series analysis.

The third test situation also tries to predict future power generation in wind parks, but includes data from neighbouring parks. This is done because information from nearby parks might improve the predictive power of the regressions by taking into account conditions that are not observable in the local wind park, such as changes in weather conditions that might affect the park at a later time.

The discussion then interprets the regressions of the various predictions. It is concluded that Linear Regression is a poor fit for predicting wind energy from wind speed, but does a better job at predicting based on history. K Nearest Neighbours does a slightly better job, but struggles with the same situations that Linear Regression does. The Machine Learning algorithms Support Vector Regression and Multi-Layer Perceptron does the best, and it is discussed how these two methods can be tinkered with to achieve even better results. However the quality of the predictions could be improved further by wind park specific tuning of parameters, and of more computing time to include more input data.

2 Introduction

Machine Learning is a field of study that has shown promise in improving the accuracy of predictions within multiple fields of study. Most have heard of the advances within artificial intelligence in recent years exemplified by such things as self-driving cars, computers that compete in game shows, or speech recognition. The improvement of the technology is quick and unpredictable. Which is exemplified by DeepMind's AlphaGo program recently beating the world champion in the board game Go. Just a few years ago, with traditional methods this was considered to be far too computationally expensive, by many orders of magnitude[6].

The technologies and methods from Machine Learning has been applied on a lot different domains and problems, but there is still a lot left to be discovered or improved upon. Machine Learning has already seen some use within Wind Energy prediction and is a growing research domain. The use of Machine Learning in wind energy prediction has the potential for saving a lot of otherwise wasted energy, because even a small improvement in predictability or usable power output would lead to huge gains due to the grand scale of the energy sector. Improving the predictability of wind energy has the effect of making wind energy cheaper to generate relative to other sources, which would help displacing more expensive or polluting methods of generating power. A discovery that improves predictability or usable energy from wind parks can also have a huge benefit for already existing wind farms. This is because the old wind parks can draw advantage of the new discoveries without having to invest in improvements of the existing wind park infrastructure, due to the improvements taking the form of a software change away from the actual wind park.

This thesis will use methods from Machine Learning to investigate how much can be gained from implementing relatively simple Machine Learning methods. The aim of this thesis is to answer the following question: Can the methods from Machine Learning improve our ability to predict energy generation from wind parks over older methods?

In order to discuss the thesis question this thesis needs to specify which set of algorithms to compare. Algorithms previously used to generate predictions as well as algorithms from Machine Learning needs to be included so that the effectiveness of implementing Machine Learning can be determined. The thesis will use the following algorithms for Machine Learning: Multi-layer Perceptron Regression, and Support Vector Machine Regression. For traditional methods I have used Linear Regression, and K Nearest Neighbours Regression. Thus the thesis will determine if Linear Regression and K Nearest Neighbours can give a satisfactory prediction of future power generation relative to some Machine Learning Algorithms.

The thesis question first be explored by first looking at some older methods for predicting energy generation, and then doing comparable analyses on the same data using a couple of Machine Learning algorithms. This will be done by generating regressions using Linear Regression and K Nearest Neighbours and then repeating the analyses using Support Vector Regression and Multilevel Perceptron Regression.

To adequately discuss the central thesis question this thesis is structured in the following fashion. First a discussion about the source and contents of the data used in the thesis. Then a presentation of the various tools and libraries used in the creation and handling of the data. Following this there is a short introduction to each of the four methods used to generate predictions. As well as some information regarding the how the data is supplied to the algorithms, and the use of error metrics for the regressions. Then the thesis moves on to show some regressions resulting from tasking the algorithms with predicting power generation from a subset of the data containing only wind speed. Then the thesis shifts focus to the regressions generated by using the algorithms to predict future power generation. The thesis then discusses the effectiveness of varying input settings for one of the Machine Learning algorithms, Multi-Layer Perceptron, and value that can be gained by tuning it correctly. The thesis then concludes on the effectiveness of the Machine Learning algorithms relative to the traditional methods.

3 Data - The Western Wind Integration Data Set

All graphs and analysis in this thesis is done on the data-set known as The Western Wind Integration Data Set, hereafter referred to as the Western Wind data-set. The data-set consists of a set of locations of thought to be potential locations for wind parks across the western half of The United States of America.

Each of the locations have for the years of the study (2004-2006) data about wind speed, and power generation retroactively estimated using data from other sources, such as weather data.

3.1 Creation of the data-set

The Western Wind data-set was created as part of and for use in the Western Wind and Solar Integration Study (WWSIS)[4].The study was organized by the National Renewable Energy Laboratory (NREL), which is a research laboratory under the U.S. Department of Energy (DOE)[3]. The goal of the WWSIS study was to determine if a large amount of wind and solar energy could be effectively integrated in the power distribution network of the western United States of America[8]. To determine this they created several data-sets to do their analyses on. One of these data-sets is the Western Wind which is used in this thesis.

3.2 Size and contents

The data in the Western Wind consists of 32043 different locations across the western United States for possible positions of wind parks. These wind parks are assumed to hold ten VestasV90 wind turbines generating up to 3 megawatt (MW) of energy each, for a total max output of 30 MW for that wind park. The wind parks are generally placed in a grid with each wind park being about two kilometers away from the ones next to it[4]. The 32043 wind park locations have data for the years 2004, 2005 and 2006 in ten minute increments leading to 157823 wind speed, and 157823 wind power data-points for each wind park. The wind power is measured in megawatts ranging from 0 to 30. The wind speed is measured in meters per second, and has no set range, however power generation from the wind parks in the Western Wind data-set can generally be seen to stop at around 26 meters per second. This matches well with the official information brochure listing 25 meters per second as the cut-of point[7].

3.2.1 Turbine Cutoff

The turbines used in the wind parks in the Western Wind data-set are a set of ten VestasV90 3 Megawatt Turbines[4]. These turbines like all wind turbines have an upper limit to how strong winds they can operate in. The data from the Western Wind data-set shows that they cut out and stop generating energy somewhere around 25 meters per second. After they have stopped like this they can not start up again (cut-in) unless the wind has dropped significantly below this level. Cut-in can be seen to happen at around 20 meters per second, which matches the official information broshure which lists 20 meters per second as cut-in speed[7]. The frequency of cut-ins and cut-outs varies a lot by location, and the usual winds speeds for that location. Some locations cut out rarely, while others cut out relatively often.

3.3 Normalization and Extra Preprocessing

The power generation data found in the Western Wind data-set is already normalized between 0 and 30, which matches the actual power generation of the wind park in megawatts. For doing statistical analysis it is more convenient to have data range from zero to one. The wind has no upper bound, but the relevance of the data falls a lot once it passes 30 meters per second.

Power generation from the wind parks were already normalized to be from zero to 30 in the original data-set, but for this thesis it was decided to normalize the data between the ranges of zero to one instead. When used the wind speed has not been normalized, but graphs may sometimes cut out some data since all there is very little relevant data above around a wind speed of 40 meters per second.

3.4 Feature Window and Prediction Distance

A parameter specifying which data a method is looking at when creating prediction is needed. In this thesis it will be referred to as that method's feature window, and the prediction distance. These two parameters can when combined describe how much data is used to generate predictions, and describe the size of the gap between the data used for predicting and the data-point that is being predicted. The figures on page 9 illustrates how the feature window, and prediction distance relate to each other. The figures contains a time series for a single wind park with eight data-points. The data points covered by the window size are used by the regression algorithms in order to predict the data selected by the prediction distance. On page 9 there is also an illustration for how the feature window and prediction distance behaves when there are more than one wind park in the input data.

Larger feature windows and prediction distances does reduce the amount of predictable values, based on how far apart the earliest and latest data used. In the illustrations there are eight time steps, but only the later four can be predicted when using a feature window of four and prediction distance of one. This is not a major problem when dealing with the Western Wind data-set due to its size, but it does mean graphs created with different feature windows and prediction distances will look shifted relative to each other when rendered as graphs.

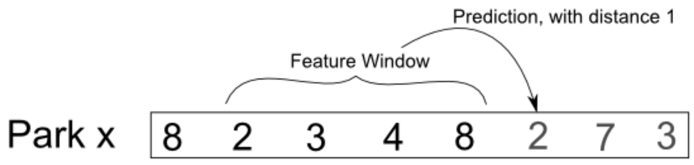

Figure 1: Four values predicting the next value

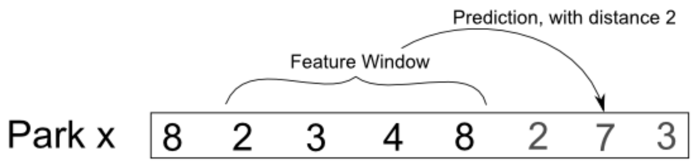

Figure 2: Four values predicting a value two steps later

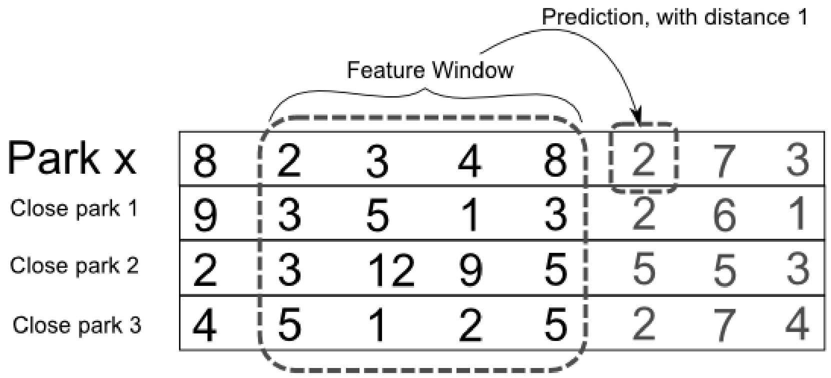

Figure 3: Four times four values predicting the next value for one park

3.5 Training and Testing Data

To determine the quality of the regressions created later in the thesis a test data-set is needed. However if the same data-set is used for 'training' the Machine Learning algorithms and for testing we can get some problems. Mainly that the

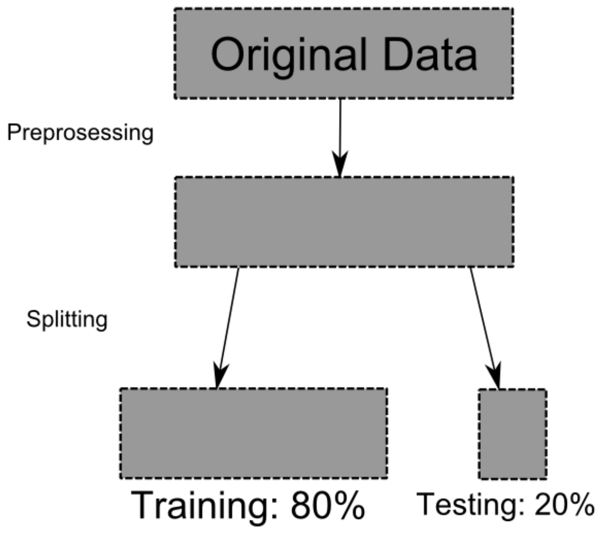

Figure 4: Overview of data handling

true quality of the regression can be hidden by the regression being overfitted. When a data-set is overfitted it is too closely matched and/or tuned to the data-set used to create it, and can give bad results when presented with new data[1, p. 19].

To avoid the problem with overfitting the data used for this thesis has been split into two distinct data-sets. For use as training and testing data respectively. The data is split chronologically to preserve information related to ordering, which is relevant for predictions in chapter five and six. Unless otherwise noted the chronologically first 80% of the data is used for training, while the remaining 20% is used as testing data.

There are some potential problems with doing it this way, like skewing the data in unforeseen ways. For instance if the regressions were only done on the first half of a year's worth of data, the testing data would be during the summer, but the regression would have been trained on data from earlier in the year. This could be bad if the location has seasonal variations that affect predictions. Likewise if three years of training data is used the testing data would be the last year's fall, which may not be representative. These situations could lead to a reduction in prediction power, but have not been tested for.

The figure on page 10 shows the steps the data from Western Wind goes through before use.

While training data is not the proper term for the input data for the two non-Machine Learning algorithms it has been used when referring to their input data to reduce the number of terms used to describe the exact same data.

3.6 Park Selection For Graphs

Most of the graphs shown in this thesis are generated using the wind park with ID 4155. This location is in the area of an already existing wind park called Tehachapi Pass Wind Farm[10]. This location is one of the locations set up by the developers of windML as a location worth looking at by adding it to a dictionary of several wind parks in the windML library. The location is interesting for testing algorithms on because the wind is variable, without reaching extreme levels. Of particular note is the fact that the wind can reach speeds which cause the wind turbines modelled in the Western Wind data-set to cut out to prevent damage to themselves. Coupled with the turbines not cutting in again before the wind speed has dropped back down to 20 meters per second[7] means that being able to predict the power output of the turbines around these wind levels is quite useful. See the graph on page 20 for an example of how the wind parks cuts out when the wind speed goes too high.

3.7 Tools

In this thesis was done using the following tools. The programs were written in Python, using libraries from NumPy for large array handling, Matplotlib for chart generation and scikit-Learn for the Machine Learning functions. In addition I used a library called WindML which handles data import, and structures the data for easier retrieval.

3.7.1 Python

All code was written in standard Python version 2.7.X, in a virtual machine running Ubuntu 16.04. The code written should be fully portable and compatible with all systems running Python 2.7.X. The code used to generate the graphs can be found in the appendix.

3.7.2 NumPy

NumPy which is the standard way of handling large arrays in Python was used for handling the data in the anaysis. It is required by both the WindML library, as well as the scikit-learn library. NumPy is usually included with most installations of Python.

3.7.3 scikit-learn

The Scikit-learn library is a library with implementations of various Machine Learning algorithms. It also has implementations of various error metrics which can be used to determine the quality of the predictions. The scikit-learn library is available to everyone, and instructions for installation is available at their website[5].

3.7.4 Matplotlib

The MatplotLib library is a library for drawing and generating graphs of various types. The matplotlib library is available to everyone, and instructions for installation is available at the website[2].

3.7.5 WindML

WindML is a library specifically designed to experiment with Machine Learning algorithms on Wind Park data. The library has standard methods for importing data from the National Research Energy Laboratory (NREL) and structures it for use with the methods provided by the library. It relies on the use of scikit-learn. The source code is available from their github account with setup and documentation provided on their website.[9]

Setup is relatively simple with only a clone of the github repository, and a specification of the root of the downloaded folder added to the operating system's PYTHONPATH environment variable.

4 Prediction Methods

In order to determine if the Machine Learning methods are better than the older traditional methods, we need to generate regressions using all the algorithms. We also need an error metric to determine the quality of the regressions. Once we have an applied our error metric on each of the regression we can figure out the how the algorithms stack up to each other, and whether the two Machine Learning algorithms prove an improvement over the traditional methods.

The following four methods, Linear Regression, K Nearest Neighbours, Support Vector Regression, and Multi-layer Perceptron regression, will all be used on three distinct situations. The first situation is for predicting power generation from wind speed data. The second is predicting future power generation using time-line analysis for a single location. The third is predicting future power generation using data from several nearby locations.

The four methods were all generated using one set of data, and tested on a second set of data. This was done in order to detect if the whether the algorithms were generating too complicated regressions which fit the input data too closely and losing its predictive ability on new data. This is called overfitting within Machine Learning and statistical analysis[1, p. 19]. For a complete explanation about input and testing data check section 3.5 in the data chapter called "Training and Testing Data". The first set of data will generally be referred to as the training data. The idea behind calling it training data is that the Machine Learning algorithms will look at this data in order to 'train' itself to recognize correct results. In order to avoid having to make a distention between 'input data' and 'training data', which both refers to the same data-set, both will be referred to as 'training data'. This means the data used to generate Linear Regression and K Nearest Neighbours regression will have their input data called training data, even though these algorithms does not really train the same way the Machine Learning algorithms does.

4.1 Error Metrics

There are several possible error metrics, but the regressions generated by these algorithms will only have the Mean Squared Error metric displayed, next to them. The Error metric will be the primary way of deciding the quality of the prediction.

4.1.1 Mean Squared Error

The primary error metric used in this thesis. It is also known as the Mean Squared Deviation. Mean Squared Error is an error metric that takes the average (mean) value of the squared (prediction) error values. As an error metric this means that errors that are further from the target values are more important for determining the quality of our predictions. If we have nine predictions that are exactly right, but the tenth are very wrong, the information about the tenth being wrong is more relevant for describing the quality of the set of ten predictions.

4.1.2 Mean Squared Error Example

Suppose one of the algorithms generated the following four predictions for wind power. 23, 25, 19, and 13, but the actual values were found to be 21, 21, 15 and 13. The differences between the predictions and the results would be the amount by which our prediction was wrong. This is called the prediction error of the prediction. To calculate the error we take 23 minus 21, 25 minus 21, 19 minus 15 and 13 minus 13 giving us the error values of 2, 4, 4 and 0.

To go from these values to the Mean Square Error we square each of the prediction errors, and find the average of the squared values. The squared values are calculated to be 4, 16, 16 and 0. We now take the average of the values which is (4 + 16 + 16 + 0) / 4 giving us the Mean Square Error of 9. Note how the larger errors of the second and third value have a greater impact on the final Mean Square Error value.

When finding errors in this thesis a prediction is generated by the regression, and tested versus a test data-set generating the initial prediction error values for each prediction. The Mean Square Error is generated using these prediction errors, and is the error value shown, even if the labels simply state ”error:”.

Since wind power generation data is normalized between zero and one in this thesis most of the Mean Squared Error values will be small. The errors will be from 0.0356 for a very bad value, down to 0.002544 for the best predictions.

4.1.3 Root Mean Squared Error

Root Mean Squared Error is not reported along with the graphs. It is simply the square root of the Mean Square error. It is the error that gives the range of one standard deviation for the prediction errors. Roughly 68% of prediction errors fall within this range on unspecified data-sets. The highest RMSE in the thesis is 0.18868, and the lowest is 0,050439, on predictions ranging from zero to one.

4.2 Algorithms

These are the two algorithms used to create regressions that are not Machine Learning algorithms.

4.2.1 Linear Regression

The thesis have used the implementation found in in the scikit-learn Python library. Linear Regression is a simple method for creating regressions. It tries to find a line through the data-set which minimizes the Least Squares error value[1, pp. 64-66]. It gives its answer in the form of a line, in a number of dimensions equal to the number of dimensions of the input data. The algorithm is provided with the chosen input data, and target values. It uses these to generate a regression line through the many-dimensional input line that best matches the data. The resulting regression is then tested new values, and a Mean Square Error is generated.

4.2.2 K Nearest Neighbours Regression

The thesis have used the implementation found in the scikit-learn Python library, where it is known as a KNeighboursRegressor. K Nearest Neighbours regression is a simple method for creating regressions that aren't linear. The algorithm looks at the average for nearby values. On page 21 an example can be seen of K Nearest Neighbours regression with K set to 25. The jagged line is the line of created by generating an average of the 25 values closest input features. In chapter 5 this is would equate to the immediate right and left on the graph, but in chapter 6 and 7 the K Nearest Neighbours regressor will look for neighbors in a higher number of dimensions. The line in the mentioned graph is pretty jagged in some areas because the 25 closest values sometimes being zero, and sometimes a value closer to 1.

The averages is then used as a regression, which is tested on new values, and a Mean Square Error is generated.

4.3 Machine Learning Algorithms

These two algorithms are used to create regressions, and fall within the domain of Machine Learning. These two methods, like most Machine Learning algorithms make use of the training data by learning as they go through the data sequentially. Learning rate and error metrics can be adjusted in a lot of different ways. For example by setting a flat learning rate for new data, or by adjusting how much weight to give new information provided to the algorithm as it gets further and further into the data-set they work on. There is a myriad of ways to pair the algorithms within the domain of Machine Learning with various error metrics, and initialization methods, some may focus on fine-tuning, while others are chosen for ease of computation. The ease of computation is sometimes desirable in cases where large data-sets are used

The two algorithms used here will both be from the group of Machine Learning algorithms called Supervised Learning, which is a set of Machine Learning algorithms where we attempt to have the algorithms learn from a set of examples in an attempt to try to get them to generalize the correct solution to the problem[1, p. 6].

4.3.1 Support Vector Regression

The thesis have used the implementation found in the scikit-learn Python library, where it is known as an Epsilon-Support Vector Regression or SVR. Support Vector regression is based on Support Vector machines, which are a kind of Machine Learning algorithm for classifying data by finding a linear separation with the biggest biggest separation between two classes. For classifications problems that aren't linearly separable the Support Vector Machines uses various methods, known as kernels to force the data into having more dimensions than the input data. It then looks for a linear separation through this new higher dimensional data[1, pp. 176-178]. The kernel used in this thesis is known as the Radial Basis function kernel, this is the default in the scikit-learn function.

When using the modifying the Support Vector Machine to a Support Vector Machine Regressor, it uses the Least Squares Algorithm internally, and needs to be supplied with an epsilon value [1, p. 186]. This epsilon value describes a band next to the regression for which the Support Vector Regression considers any data points within as good enough. The internal error metric only looks at values outside of this band.

The Support Vector Regression can be very slow, and without doing any trimming of the data, training the Support Vector Regression takes some time, when training on the complete time-line for one or more parks in the Western Wind data set. Increasingly so when more input data is added in Chapter 6 and 7.

4.3.2 Multi-layer Perceptron Regression

The thesis have used the implementation found in the scikit-learn Python library, where it is known as Multi-layer Perceptron regressor or MLPRegressor.

The Multi-Layer Perceptron is based on the Perceptron. The Perceptron is an Machine Learning algorithm that uses two layers of node-like objects, referred to as neurons, to generate its output. The neurons have an activation function which determines its output based on the inputs given to the neuron. The inputs from different other neurons, or input layer, have weights assigned to them. These weights determine how much the neuron bases its decision on each of its inputs. The learning in the Perceptron functions by clever adjustment of the weights. This can be seen by looking at the algorithm for training the Multi-Layer Perceptron being focused on updating the internal weights only making changes to the internal weights. Pseudocode for the Multi-Layer Perceptron training can be seen in *Machine Learning - An Algorithmic Perspective*[1, p. 78].

When training the system a prediction is made, and the weights are adjusted based on the size of the error.

The neurons in a Perceptron or a Multi-Level Perceptron are ordered in layers with every single node in one layer, is connected to every single node in the next. An illustration can be seen on page 17. Each of connections, shown here as arrows, between nodes have a weight for how much one neuron affects the next.

The Perceptron with its single layer is limited in how complex predictions it can learn. It can only solve problems that are Linearly Separable[1, p. 55]. Adding another layer of nodes to the network allows a Multi-Layer Perceptrons to learn more complex problems. This is because the knowledge generated in one layer can be used as input for the next, as illustrated in the book *Machine Learning - An Algorithmic Perspective*[1, pp. 86-87].

In this thesis the weights inside the Multi-Layer Perceptron is set to be small random numbers. This random initialization is done so that the value of each neuron does not overlap completely with another. Such overlap inhibits the neurons ability to discover distinct patterns in the data.

The random values assigned to the internal weights are are small, but too far from each other in order for them to reach their final values about the same time in the training[1, p. 80].

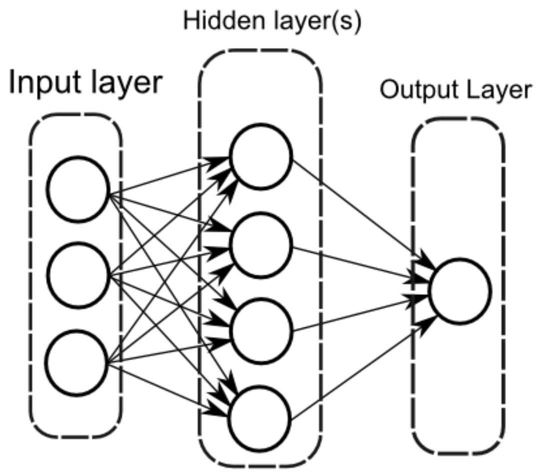

Figure 5: placement of nodes in a multilayer perceptron

4.4 Training and Testing Methodology

I train on the training data, then I test on the testing data. The training and testing data will be the same for each algorithm, and should be directly comparable quality wise. This mimics a how it would work in real life situations under normal conditions, because we can then get the result of which algorithm best predicts a the power generation of a specific wind park.

5 Estimating Power Generation Using Current Wind Speed

While the Western Wind data-set supplies both power generation and wind speed for each location at all times, having the ability to independently predict the power generation from wind speed data is useful. Creating power generation forecasts using only wind speed information could allow for the use of Machine Learning on data created from weather forecasting. These predictions would be degraded in relation to how close the weather forecasting is to the actual weather it predicts, but could could, given accurate forecasts make estimates that are good enough to plan around.

In these following four groups of regressions I have used wind data from location 4155 from the years 2004 to 2006. Each of the regression methods have been trained on the first 80% of the data, with the remaining 20% being used for testing, and is shown in the graphs. The error value is generated as the Mean Square Error generated from the prediction error between the generated regression, and the testing data.

5.1 Extra Preprossessing

None. The MLPR does not handle the lack of preprocessing well. MLPR did better when wind parks that had been cut off due to high wind speeds had been removed as outliers, but no graphs for that situation has been included.

5.2 Linear Regression

The graph on page 20 illustrates Linear Regression predicting the power output from park 4155 by using wind speed as input data. The regression tracks the power generated pretty poorly with a Mean Square Error of 0.0356. Linear Regression has particular trouble generating a decent result due to only being able to create straight lines, while the optimal curve would need to be curved to follow the data closely.

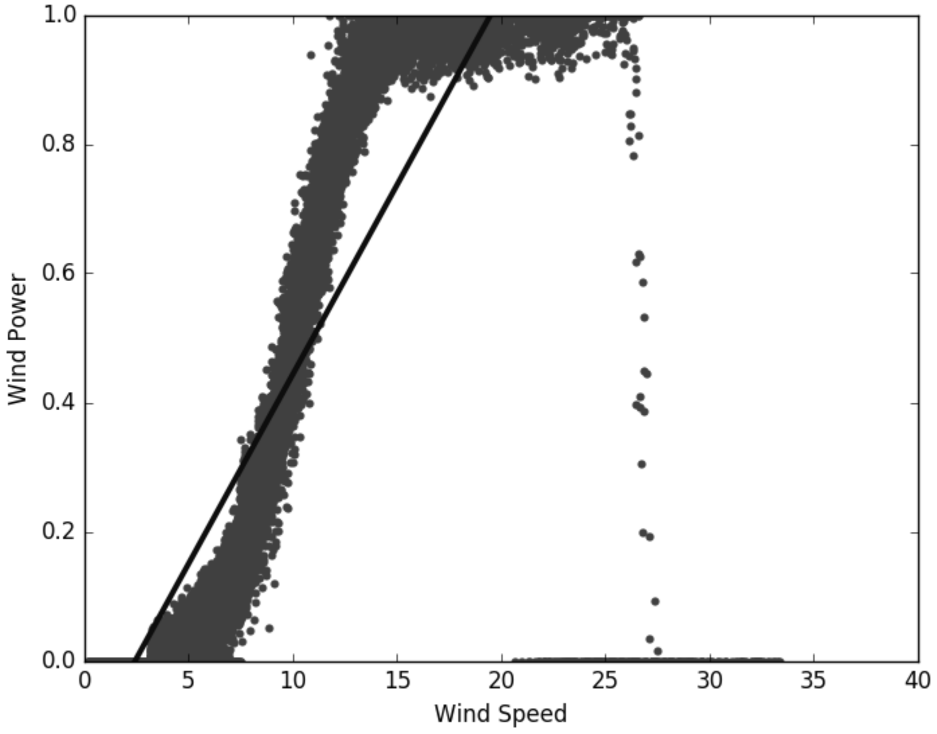

Figure 6: Linear Regression in black on grey testing data. Park 4155. Error: 0.0356

5.3 K Nearest Neighbours Regression

The graph on page 21 shows a K Nearest Neighbours regression predicting the power output from park 4155 by using wind speed as input data. K was set to be 25. This is the same data as in the Linear Regression above. The KNN regression is tracking the power curve far better than the linear regression, with a Mean Square Error of 0.00417. The graph looks ugly with a lot of jagged edges in part because the times where the wind park has cut out due to previously having too high wind speed, but the wind hasn't yet dropped down to a level where the wind park can start up again. The next graph runs the

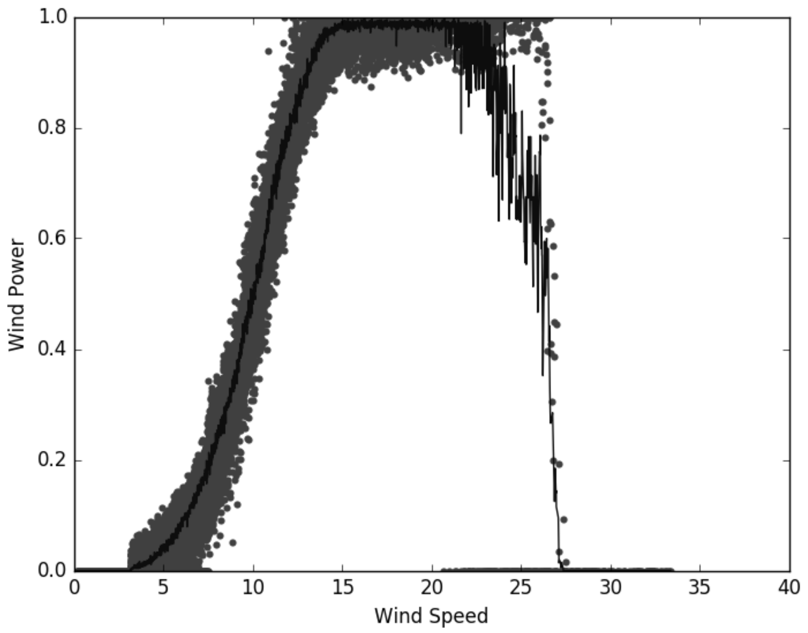

Figure 7: K Nearest Neighbours Regression in black. Park 4155. K=25. Error: 0.00417

same regression with a higher K value of 500 in an attempt to get rid of the jaggedness. This makes the look nicer graph nicer, but the predictive power is largely unchanged with a Mean Square Error of 0.00403.

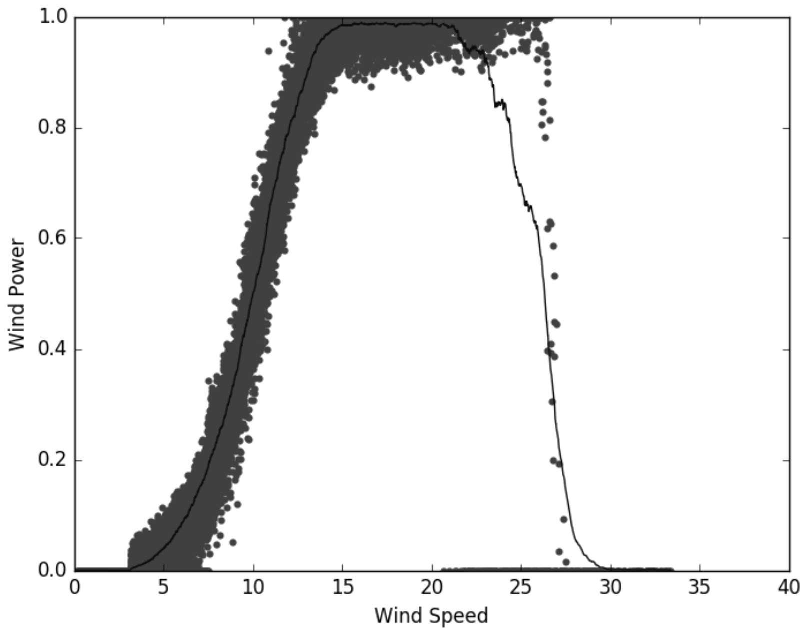

Figure 8: K Nearest Neighbours regression in black. Park 4155. K=500. Error: 0.00403

5.4 Support Vector Regression

The graph on page 23 shows a Support Vector regression predicting the power output from park 4155 by using wind speed as input data. The epsilon value was set to be 0.01, this is the range for which the regression attempts to put the data inside of. The data for this regression was reduced by four fifths for performance reasons. Every fifth data-point was used. The graph tracks the power curve decently, and finds the curve that would intuitively be more correct by weighting the zero values between 22 and 27 meters per second, less than staying close to the power curve of the active wind parks. The Mean Square Error was found to be 0.00499

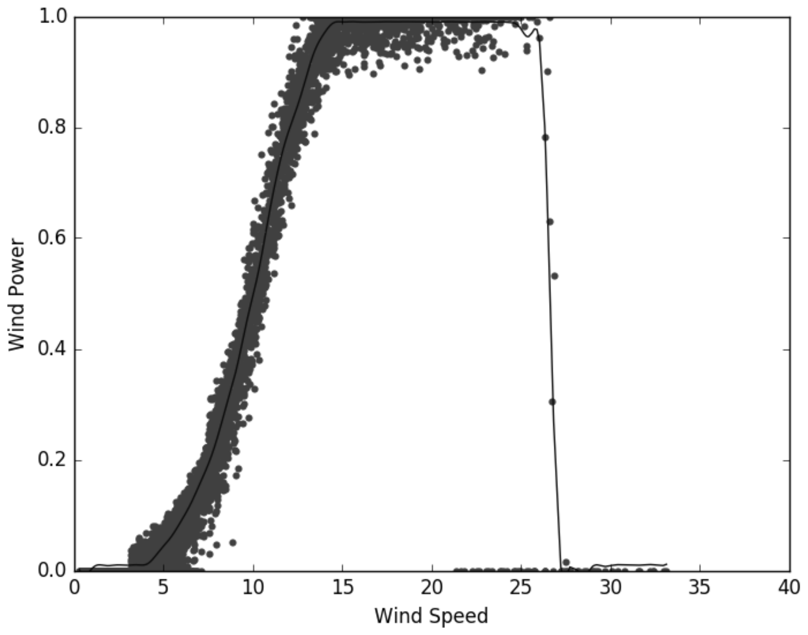

Figure 9: Support Vector Regression in black. Data size reduced to 1/5 for time. Park 4155. Epsilon=0.01. Error: 0.00499

5.5 Multi-Layer Perceptron Regression

The graph on page 24 shows a Multi-Layer Perceptron regression predicting the power output from park 4155 by using wind speed as input data. A single hidden layer with 300 nodes was used. Like the K Nearest Neighbours regressor the MLPR struggles with dealing with the values between 22 and 27 meters per second, and ends up generating an average for the area instead of tracking an area with higher data-point density. The Multi-Layer Perceptron was randomly initialized, which leads to slightly different results each run. The Mean Square Error was in this regression found to be 0.00475

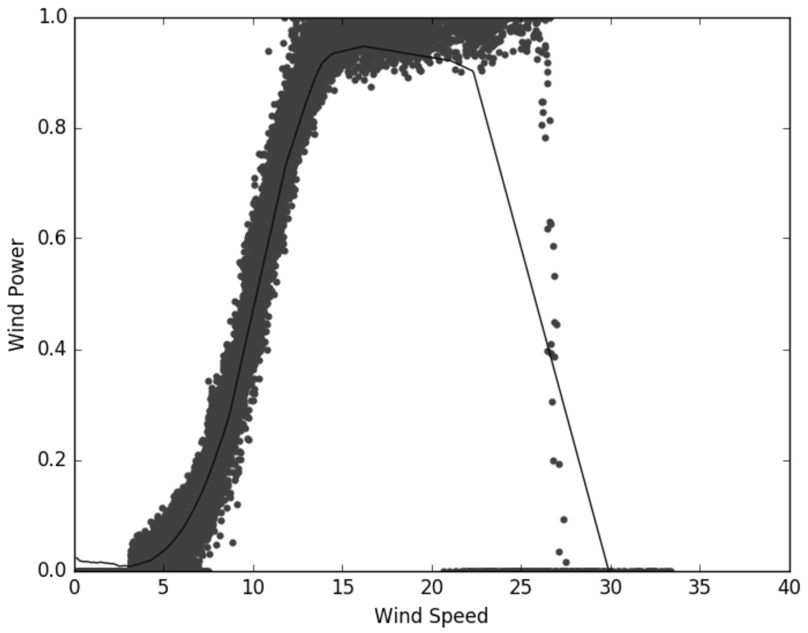

Figure 10: Multi-Layer Perceptron regression in black. Park 4155. One hidden layer of 300 nodes. Error: 0.00475

5.6 Problems With Wind Park Cutoff

As described in the data chapter wind turbines can cut in and out of operation when wind speeds become too high. A cut-out happens, followed by a cut in when the wind speed is somewhat lower. When predicting power generation using only information about current wind speed, the information about whether the turbine is currently cut out due to previously high wind speeds is entirely missing. This leads the regressions in this chapter to blend the data from wind parks with turbines that are operational with wind parks where turbines are not operational.

6 Predicting Power Generation - Time Series Analysis

As shown in the chapter about doing estimates exclusively using wind speeds and no history information, the predictions were not perfect. The predictions did not, and could not, take into account the knowledge of whether the wind park is currently shut down, either from high wind speeds, or other reasons. To take this into account we need to use at least some history information for the wind park.

This chapters will show graphs illustrating regressions created by looking at the immediately prior data about power generation. For this to work we need to decide how far back we want to look back. While it makes intuitive sense to include a lot of information about history, the value of the data is lower the further we got back in the past. It may make sense for the wind power generation in the last twenty minutes at a wind park to be a decent predictor of the power generated in the next 10, but it makes less sense to also include data about power generation three months earlier. The best range is somewhere in between, and will be discussed further in the next chapter where neighbouring wind parks will also be taken into account.

In these following four groups of regressions wind data from location 4155 from the years 2004 to 2006 has been used. Each of the regressions methods have been trained on the first 80% of the data, with the remaining 20% being used for testing. The error value is being generated as the Mean Square Error generated from the prediction error between the generated regression, and the testing data. The feature window is set as the previous three time steps (30 minutes), and the prediction distance is one, which means it will predict the power generation in 10 minutes.

Each of the regressions will be generating predictions for the entirety of the test data, which is close to 31500 data-points. Looking at this many data-points at the same time is hard, as well as not really helping understanding the results. An example of this is shown on page 26. As such the example graphs in this chapter has been adjusted to show examples of predictions along the time-line. The error values reported is the error value for the prediction of the entire time-line, not just the parts shown. The size of the test data is close to the number of wind parks in the Western Wind data-set, but this appears to be a coincidence. On page 26 there is a graph showing the complete time-line of the testing data for location 4155.

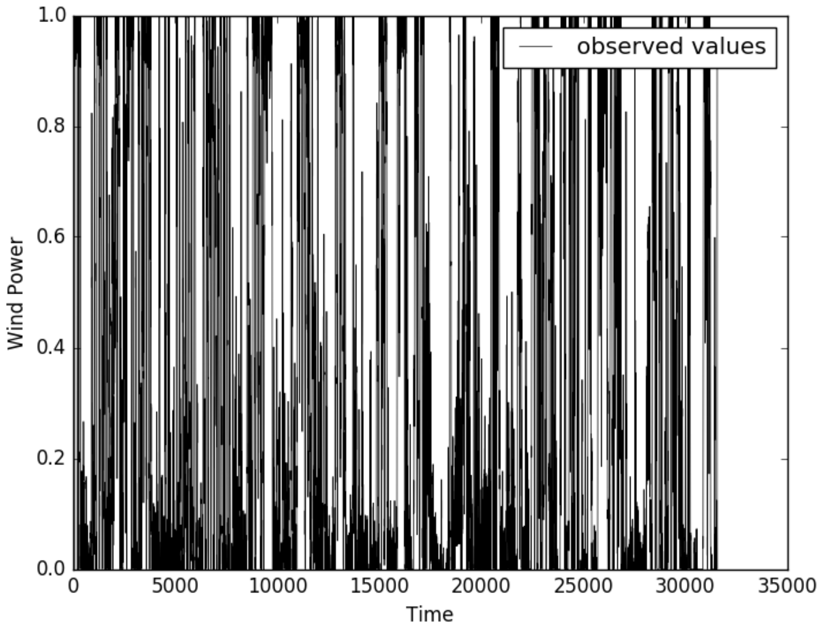

Figure 11: Full timeline. Park 4155

6.1 Linear Regression

The graph on page 27 illustrates Linear Regression predicting the power output from park 4155 by using its own previous wind energy generation as input data. The last half hour of power generation is used, with the goal of prediction power in the next time-step (10 minutes). The predictions created by the regression seem to trail the observed values, but the quality of the prediction is far better than in the previous chapter. The Mean Square Error was found to be 0.00369.

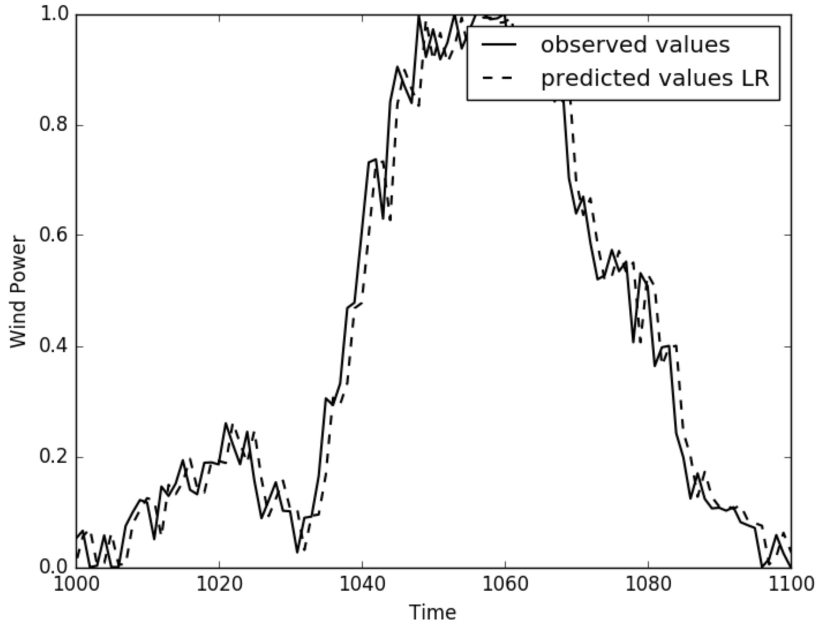

Figure 12: Prediction using Linear Regression. Park 4155. Feature window of 3. Error: 0.00369

6.2 K Nearest Neighbours

The graph on page 28 illustrates K Nearest Neighbours predicting the power output from park 4155 by using its own previous wind energy generation as input data. The last half hour of power generation is used, with the goal of prediction power in the next time-step (10 minutes). The K values was set to be 500. The prediction, like with Linear Regressions trail the observed values closely, but the actual prediction is a tiny bit better with a Mean Square Error of 0.0361

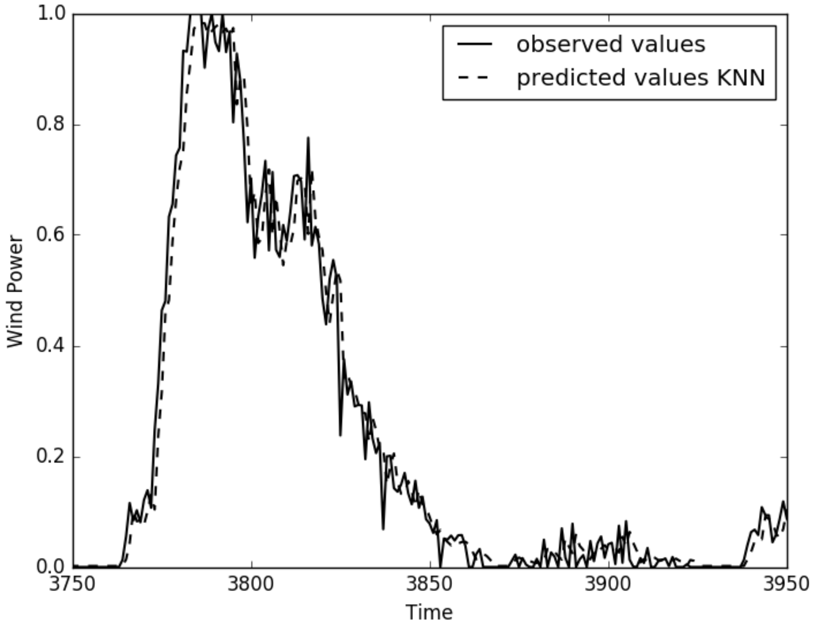

Figure 13: Prediction using K Nearest Neighbours regression. Park 4155. Feature window of 3. K is 500 Error: 0.00361

6.3 Support Vector Regression

The graph on page 29 illustrates Support Vector Regression predicting the power output from park 4155 by using its own previous wind energy generation as input data. The last half hour of power generation is used, with he goal of predicting power in the next time-step (10 minutes). The Epsilon value was set to be 0.01, which means it does not adjust based on values inside this range when training. The Support Vector Regression takes a long time to train, on this amount of data, but generated as better prediction than K Nearest Neighbours with a Mean Square Error of 0.00351.

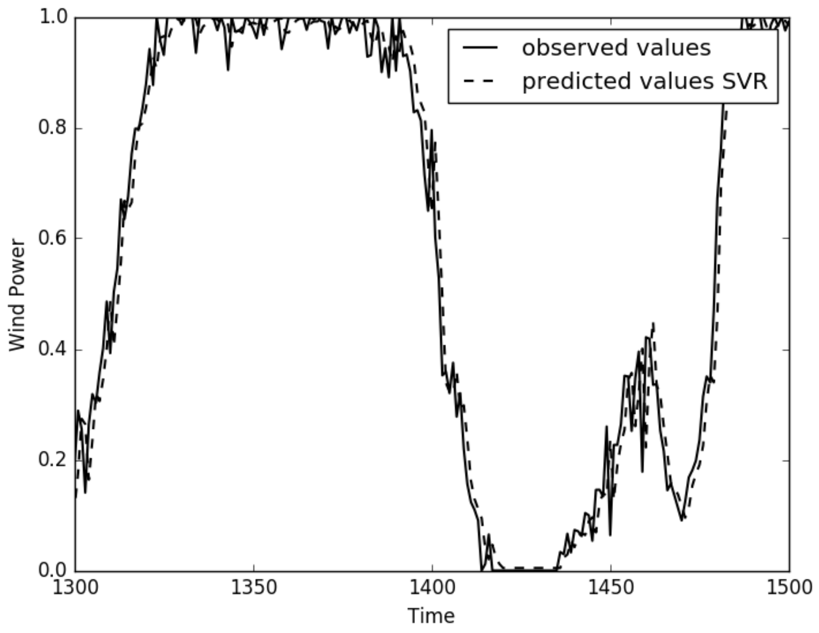

Figure 14: Prediction using Support Vector regression. Park 4155. Feature window of 3. Epsilon of 0.01. Error: 0.00351

6.4 Multi-Layer Perceptron Regression

The graph on page 30 illustrates a Multi-Layer Perceptron regression predicting the power output from park 4155 by using its own previous wind energy generation as input data. The last half hour of power generation is used, with the goal of predicting power in the next time-step (10 minutes). The Perceptron used two hidden layers of 200 neurons each. The Perceptron was randomly initialized, which leads to slightly different results each run. The Multi-Layer Perceptron did a better job than any of the other methods, but not significantly so. It's possible that there just isn't enough information present to create accurate results. The next chapter will go more into depth with the tuning parameters for the Multi-Layer-Perceptron. In this run the Mean Square Error was found to be 0.00345

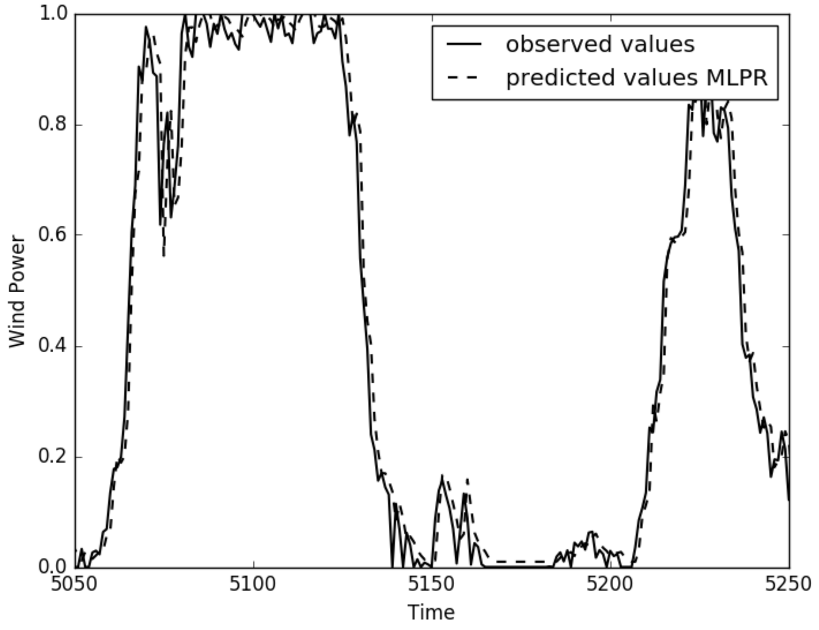

Figure 15: Prediction using Multilevel Perceptron regression. Park 4155. Feature window of 3. Two hidden layers of 200 neurons each. Error: 0.00345

7 Predicting Power Generation - Information from Local Parks

While the predictions in the previous chapter was an improvement over the ones in the chapter before it, there are still a lot of information that can be used to generate better graphs. This chapter will focus on adding power generation information from nearby parks in addition to the history data added in the previous chapter. The prediction will then be based on history data equal to the feature window of the park for which we want a predicting. As well as the history data equal to the feature window of all the parks defines as nearby. As with the size of the feature window itself, there's a drop in relevance as the parks are further and further away from the park we're trying to predict. Including other wind parks in the immediate vicinity will likely have data that are good for improving predictions. However wind parks that are far away may have little to no effect on the predictions.

Since the wind parks are roughly placed on a grid with a distance of roughly two kilometers between wind parks the it was decided that using a radius of three kilometers from the initial wind parks would include a set of wind parks sufficient to illustrate the value of the idea of including nearby wind parks. A three kilometer radius covers eight neighbouring parks if the park locations are densely packed.

In the following four groups of regressions the groups of wind data from location 4155 from the years 2004 to 2006 has been used used. Each of the regression methods have been trained on the first 80% of the data, with the remaining 20% being used for testing. The error value is being generated as the Mean Square Error generated from the prediction errors between the generated regression, and the testing data. The feature window is set to three time steps (30 minutes), and the prediction distance is one, which meant it will predict the power generation for the next time step (10 minutes). Wind parks within three kilometers, including the target wind park, will be used as input data.

Each of the regressions will be generating predictions for the entirety of the test data, which is close to 31500 data-point. Looking at this many data-points at the same time is hard, as well as not really helping understanding the results. An example of this is shown on page 26, which contains the testing data for park 4155. The error values reported is the Mean Square Error for the prediction of the entire time-line, not just the parts shown.

7.1 Linear Regression

The graph on page 32 illustrates Linear Regression predicting the power output from park 4155 by using its own previous wind energy generation as input, as well as using the wind power generation of neighbouring parks within a distance of three kilometers. For all the parks, 4155 and its neighbours, the last half hour of power generation is used, and is used to generate a prediction for the next time-step (ten minutes). The prediction created by the regression seem to be better better than the comparable Linear Regression created by only using the history data from park 4155. The Mean Square Error found when only using park the local park was 0.00369, but with the addition of the neighbouring parks the Mean Square Error was found to be 0.0278. This is a significant improvement over the one-park regression.

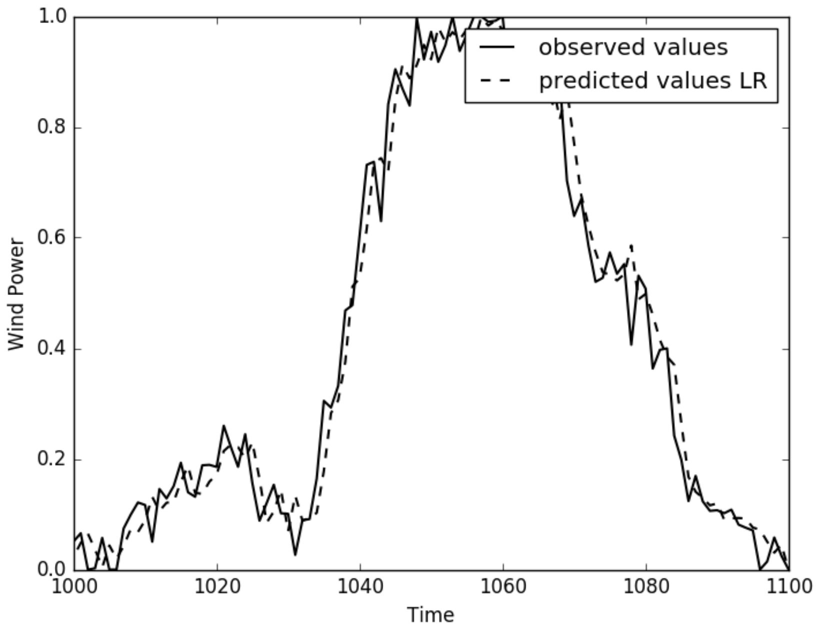

Figure 16: Prediction using Linear Regression. Park 4155. Feature window of 3, Park radius of 3km. Error: 0.00278

7.2 K Nearest Neighbours Regression

The graph on page 33 illustrates K Nearest Neighbours regression predicting the power output from park 4155 by using its own previous wind energy generation as input, as well as using the wind power generation of neighbouring wind parks within a distance of three kilometers. For all the parks, 4155 and its neighbours, the last half hour of power generation was used, and is used to generate a prediction for the next time-step (ten minutes). A K value of 500 was used. The prediction in this case does not appear to be better than the prediction generated in the previous chapter. In fact it is slightly worse with its Mean Square Error of 0.0039 vs the error from the previous chapter which was

0.00361. This suggests K Nearest Neighbours was not able to take into account the extra data from the neighbouring parks in any meaningful way.

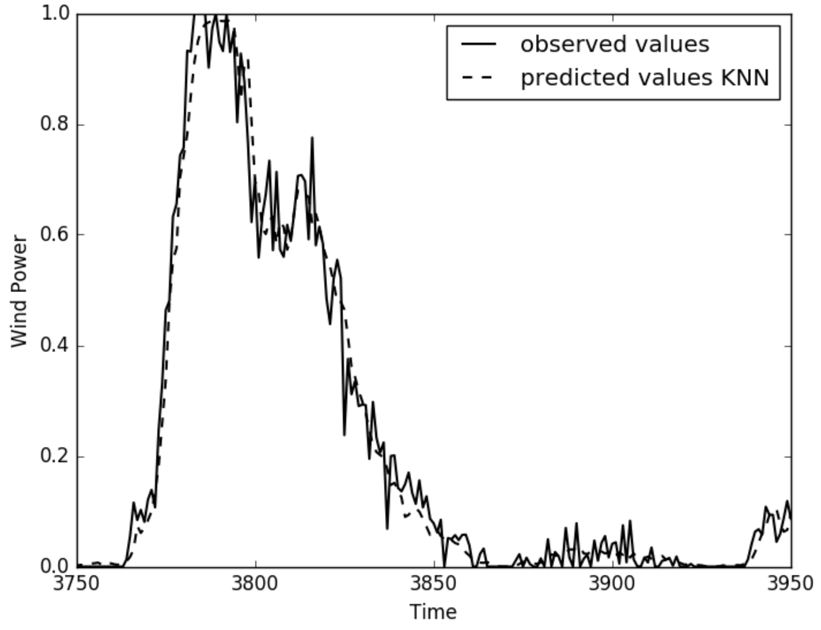

Figure 17: Prediction using K Nearest Neighbours regression. Park 4155. Feature window of 3. Park radius of 3km. K is 500. Error: 0.0039

7.3 Support Vector Regression

The graph on page 34 illustrates Support Vector Regression predicting the power output from park 4155 by using its own previous wind energy generation as input, as well as using the wind power generation of neighbouring wind parks within a distance of three kilometers. For all the parks, 4155 and its neighbours, the last half hour of power generation was used, and is used to generate a prediction for the next time-step (ten minutes). The epsilon value was set to be 0.01, which means it does not count values inside this range as errors to adjust when training on the data-set. This Support Vector Regression takes a very long time to train due to the size of data, though the Support Vector Regression could well have been trained on less data to achieve similar results. The regression shown does in this case better than both Linear Regression and K Nearest Neighbours Regression with an error of 0.0262.

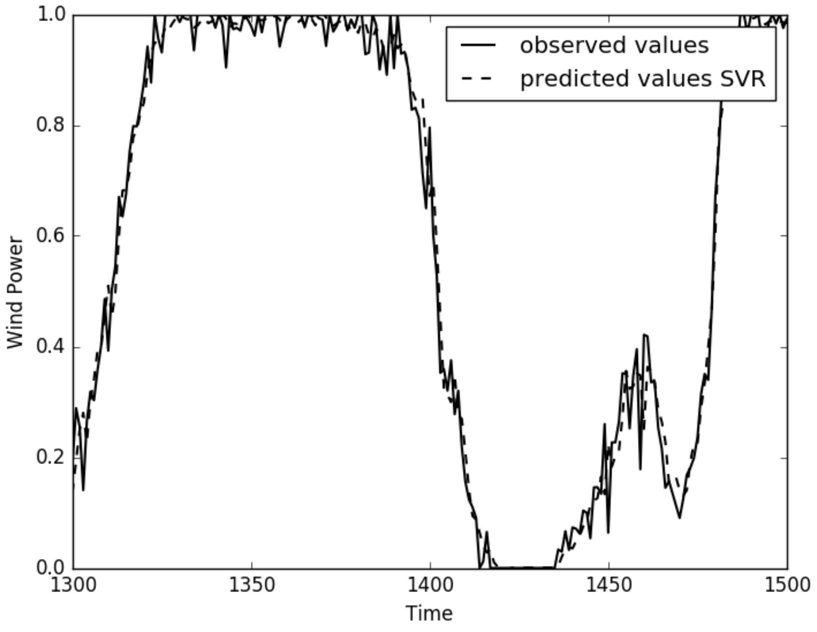

Figure 18: Prediction using Support Vector Regression. Park 4155. Feature window of 3. Park radius of 3km. Epsilon value is 0.01. Error: 0.00262

7.4 Multi-Layer Perceptron Regression

The graph on page 35 illustrates a Multi-Layer Perceptron regression predicting the power output from park 4155 by using its own previous wind angry generation as input, as well as using the wind power generation of neighbouring wind parks within a distance of three kilometers. For all the parks, 4155 and its neighbours, the last half hour of power generation was used, and is used to generate a prediction for the next time-step (ten minutes). The perceptron was set to use two hidden layers with 200 neurons each, this seemed to be enough to capture a lot of the information found in the training data, while still completing in a reasonable time. The Multi-Layer perceptron takes a shorter time to generate predictions than the Support vector regressions, which allowed for looking at some interesting tinkering with the amount of data used to generate regressions. The first regression shown, displays the results of using a prediction distance of one, a feature window of three, and to include neighbouring parks within three kilometers.

It is important to reiterate that the perceptron uses random initialization of weights between the neurons. This means that the results from each run to the next may not be the same, as the neurons will trend towards the correct solution from slightly different directions, and possibly find different local optimums.

The results of the Mult-Layer Perceptron was found to be better than any of the other regressions at a Mean Square Error of 0.002545. Though we do not know if a prediction of this quality will be generated every time the algorithm is run.

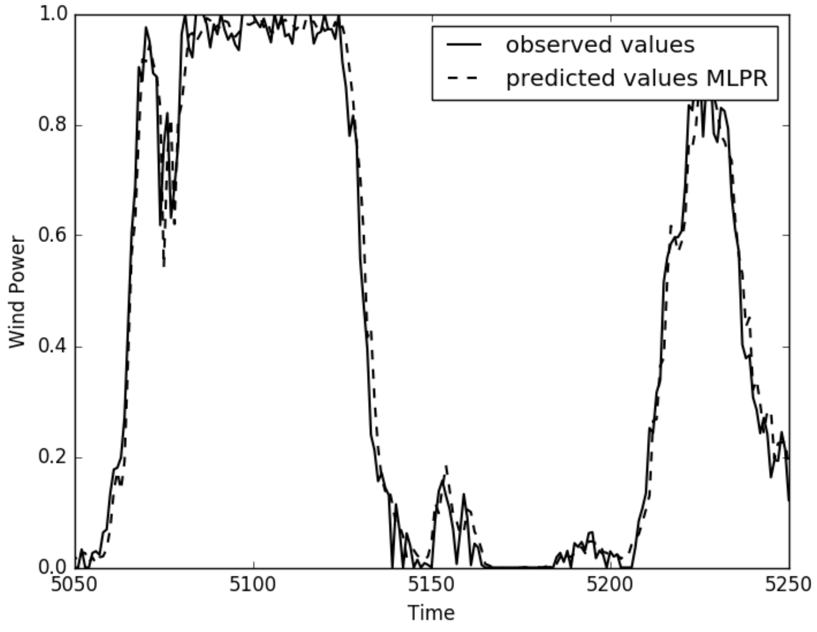

Figure 19: Prediction using Multi-Layer Perceptron Regression. Park 4155. Feature window of 3. Park Radius of 3km. two hidden layers of 200 neurons each. Error: 0.002544

7.4.1 Varying Input Settings

Since the Multi-Layer Perceptron regression takes less time to run than the Support Vector Regression we can do some tinkering with the input values. The following four graphs have checked for the effects of increasing the size of the Wind Park Radius to nine kilometers, the increase of window size to five (50 minutes or nine steps (90 minutes). All the graphs use two hidden layers of 200 neurons each.

The first graph shows the effects of increasing park radius to nine kilometers, and achieves a Mean Square Error of 0.00255. The second graph shows the effects of increasing the window size to nine (previous 90 minutes), while going back to a park radius of three kilometers. This lead to a mean Square error of 0.00274. The third graph shows the effects of increasing both window size and park radius to nine. It achieved a Mean Square Error of 0.00264. In the final graph the window size is set to be nine, while the park radius is set to a value inbetween the two previous values at five (50 minutes). This graph achieved a Mean Square Error of 0.00267.

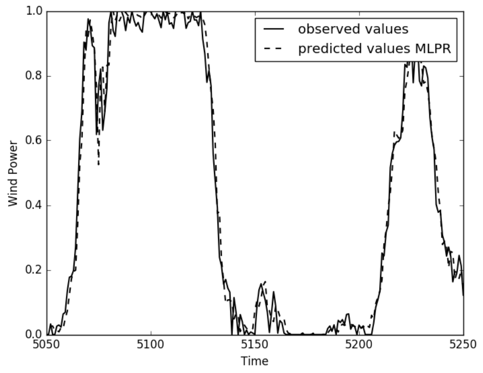

Figure 20: Window = 3. Radius = 9. Hidden layers = 2x200. Error = 0.00255

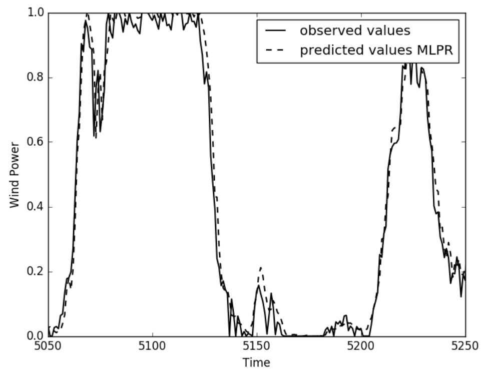

Figure 21: Window = 9. Radius = 3. Hidden layers = 2x200. Error = 0.00274

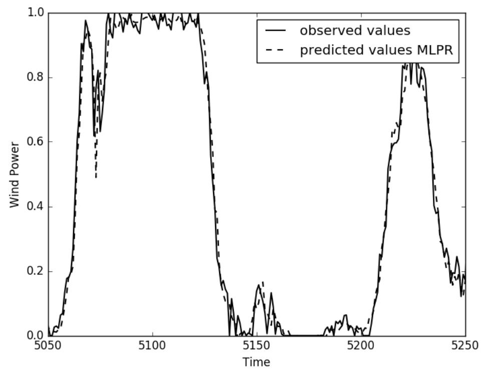

Figure 22: Window = 9. Radius = 9. Hidden layers = 2x200. Error = 0.00264

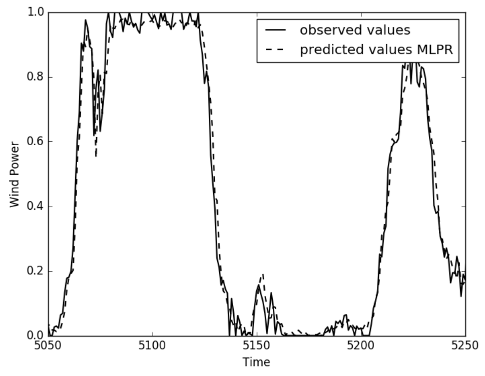

Figure 23: Window = 9. Radius = 5. Hidden layers = 2x200. Error = 0.00267

8 Discussion

The central goal of this thesis is to answer the following question: Can the methods from Machine Learning improve our ability to predict energy generation from wind parks over older methods? To be able to answer this question the thesis have created predictions using some older methods as well as predictions for some Machine Learning methods. This discussion chapter will discuss these results and findings as well as discussing the quality of the methods used in this thesis relative to those used that may have been used current or previous production environments.

The thesis will not look at algorithms that have already been fine tuned from wind park production environments, but uses approximate methods instead. These methods can be improved upon, but to which degree has not been looked upon in this thesis. The thesis will look at, and use, methods created from using the Python program library for Machine Learning called scikit-learn. This library provides standard implementation for several Machine Learning algorithms, as well as access to several traditional methods such as Linear Regression and K Nearest Neighbours. All the methods/algorithms in the library has already been set up to be accessed in a single standardized way that simplifies development and testing of the methods.

8.1 Relevance of the Data

When predicting the power generation for the wind parks it was found that some data was more relevant for the predictions than other data. Some of the data could be safely ignored, without impacting the predictive quality of the algorithms, while other parts were very important. While running the second and the third tests it was observed that using more historic data did not necessarily increase the predictive power of the regressions. It was also observed that using data from more distant wind parks behaved similarly in not directly improving the predictions. This is shown in 7.4.1 where it is illustrated on the Multi-Layer Perceptron, the algorithm deemed best at filtering out the irrelevant data, because of its system of updating internal weighting. This decrease in relevance happened for all the algorithms, but illustrations for the other regression methods have not been included.

8.1.1 The First Test - Power Generation from Current Wind Speed

When creating regressions in chapter 5 about predicting the power generation it became clear from the graphs that a regression based only on wind speed data would not generate a satisfactory result, because of the problem with wind parks shutting down in too high wind speeds. These parks would not start generating power again before the wind speed dropped significantly due to engineering constraints. The data used, which was only the wind speed in the current time step, could not provide enough information to discern whether the turbine was in operation or had been shut down due to previously high wind speeds. This can be seen in the graph on page 39, duplicated from chapter 5, where the regression line is trying to figure out the regression between findings close to one and exactly zero, which represents a wind park with offline turbines.

While not shown in the graphs it was discovered that by cutting out the zero values from about 20 to 27 meters/second the predictions could generate better predictions than when the zero values were included. This, however, would necessitate a separate way to track whether the wind parks were about to turn off or on, to avoid the data from becoming misleading. The results of these regressions could when the zero values were handled separately function as a simple mechanism for predicting power generation in production systems.

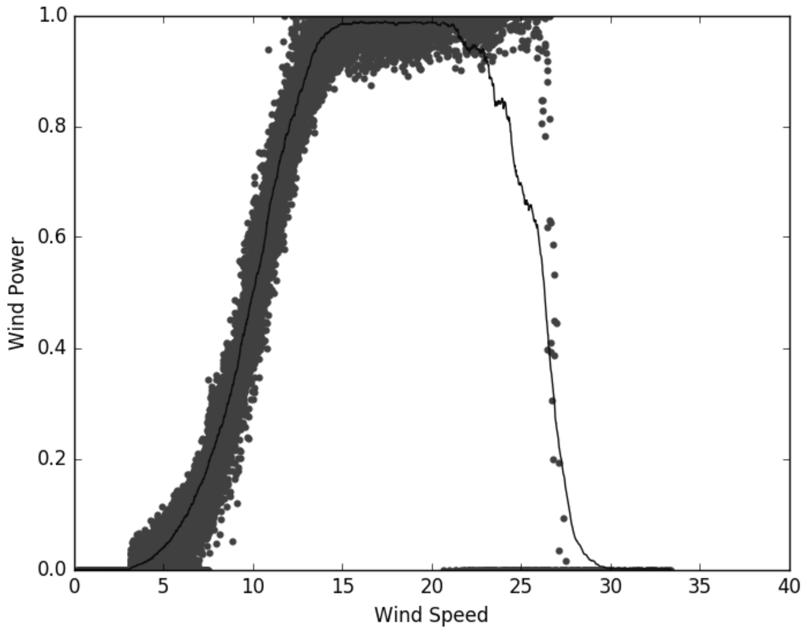

Figure 24: KNN regression. Park 4155. K=500. Error: 0.00403

8.1.2 The Second Test - Power Generation From Recent Power Generation

In the second test the input data of wind speed was replaced with data containing wind power generation for the given wind park for multiple time steps. This means that unlike in test one, it was possible for algorithms to know if the wind parks were turned off or on immediately before the time we are predicting for. This means it's possible to know the status of the wind park for a period of time before the time we are predicting for. This has clear advantages relative to the first test, because it allows the algorithms to correctly handle situations where the turbines in the wind parks are turning off and on, as well as knowing the current state of the wind park, with their set of ten turbines.

The most important piece of data provided from the wind park history is the most recent power generation reading. This piece of data contains the information that allows us to determine whether the wind park is active or not. While creating the regressions it was observed that as the data reached further into the past its relevance decreased significantly, because how the weather changes tends to be affected most strongly by recent trends.

The graphs in chapter 7.4.1 compares the use of the three most recent data-points (30 minutes), relative to the use of the nine most recent data-points (90 minutes) for predicting power generation. The 90 minute prediction was actually worse than the 30 minute prediction, but only slightly. With a small difference in predictive quality a Mean Square Error of 0.002738 for the 90 minute window and a Mean Square Error of 0.002544 for the 30 minute feature window. This is likely in part due to how the Multi-Layer Perceptron has some randomly assigned starting settings, and does not come to the exact some result every time. Repeated generations of the graphs did not show any significant deviations from the provided graphs.

This means the inclusion of history data improves the predictions relative to only using the wind speed for predicting power generation, but since the relevance of the data decreases it has no practical value past a certain point. While not directly shown in the graph it seemed this point was somewhere around 30 to 50 minutes of history data.

8.1.3 The Third Test - Power Generation from Recent Power Generation of Nearby Parks

Using the same principle as in test two, which added wind power generation information from the near past, test three looks at the inclusion of the inclusion of information from nearby wind parks.

The graphs in chapter 7.4.1 compares the use of the wind parks within a radius of three kilometers, versus using a greater radius of up to nine kilometers. Similar to the results in test two this thesis found that the inclusion of more wind parks did not improve the quality of the predictions further out than a few kilometers. When the range was set to three kilometers the Mean Square Error was found to be 0.002544, while when the range was set to nine kilometers the Mean Square Error was found to slightly higher at 0.00274. The Multi-Layer Perceptron does not generate exactly the same results each time, and these two values are close enough to each other to be the cause of random variation, but it seems that the inclusion of more data did not improve the quality of the predictions.

When it comes to deciding the range of local wind parks to use there are several variables that impact how much we can get out of using the data from distant wind parks. One of them is that the computations may take a long time to complete, if the remote parks are simply included in the data-set. The impact of this is changes based on the algorithm, but some handle such increases better than others. In the case of the Multi-Layer perceptron, one of the algorithms that are good at reducing the weight of irrelevant variables. The number of neurons used to generate the regressions may have been to few. This would lead to each neuron attempting to capture too much information at the same time, and as a result of this not being able to create an estimate as good as if there were less data per neuron. According to the book *Machine Learning - An Algorithmic Perspective* there is no good way to know how many neurons are needed when creating the Multi-Layer Perceptron, only that it has to be found by trial and error[1, p. 86]. When creating the regressions in this thesis it was found that more neurons were likely needed, but this impacted the generation time significantly.

There are some aspects to as to how predictive wind power generation in one location affects the wind power generation in another. One of these is how the changes in wind speed moves across the land. An increase in wind speed found in one wind park, will only affect the next when the wind has traversed the distance between the two parks. As well as the wind direction. By looking at high wind speeds such as 25 meters per second, the highest wind speed where the turbines inside the wind parks are operational[7], it can be shown that within a single 10 minute time-step, the wind can reach from outside the furthest measuring point, to reaching and passing the wind park we are creating a prediction for. This can be seen by calculating that 25 meters per second equals 90 kilometers per hour, and the 9 kilometers radius used in the third test will be too short a distance to take this into account since these 9 kilometers takes only 6 minutes to blow from the edge of the sensing radius to the center of the wind park.

We can see that that going from a radius of three kilometers to nine kilometers could theoretically give an improvement, but the regressions used in this thesis could not find an improvement, and instead found a slight worsening. This could likely be from the particular settings used, either due to the number of neurons used in the Multi-Layer Perceptron, or due to the nine kilometer range being too small to capture changes in wind speeds in time.

8.1.4 Including Wind Speed as Separate Set of Data

The Western Wind data-set provides both wind speed and power generation data for all the wind parks from 2004 to 2006, but in this thesis only the wind power generation data was used in chapter six and seven. These were the chapters that used historical data, as opposed to trying to generate a prediction for the same time-step.

The wind speed data was not used to generate these regressions. This is in part because the inclusion of wind speed data would increase the size of the data-set by a factor of two. This increase in data-set size would then in turn increase the generation time for the regressions, for most of them by more than double. In addition to this, the inclusion of the wind speed data would likely not improve the quality of the predictions significantly, because the information conveyed are already partially covered by the wind energy generation data. The wind energy generation data was generated from several data sources, one of which was the wind speed data accessible in the Western Wind data-set[4].

Keeping in mind the increase in the data-set size it was determined that it would be preferable to attempt the inclusion of a higher number of nearby wind parks, instead of adding more information for a lower number of wind parks.

8.2 The Methods' Ability to Handle the Data

This section will discuss how well the various methods managed to create predictions for the three tests, as well as how the methods' abilities to create predictions are affected by the increase in input data. The first two methods are the traditional non-Machine Learning algorithms, Linear Regression, and K Nearest Neighbours regression. Followed by a discussion of the performance of two Machine Learning algorithms, Support Vector Regression and Multi-Layer Perceptron Regression.

8.2.1 Linear Regression

The Linear Regression struggles particularly bad with test one where the goal was to predict power generation using only wind speed. With a Mean Square Error of 0.0356 the error was significantly greater than for all the other regressions. The size of the Mean Square Error in test one is large enough to make the results entirely unusable. The cause of the large error is mainly due to the Linear Regression generating its regression as a straight line, something which is a poor fit for the data

For test two with prediction based on historic wind power generation data, the linear regression does a decent job at generating predictions, and the results appears close to accurately predicting the power generation in the next step. The Mean Square Error was found to be 0.00369.

In test three where both history data and data from nearby parks were used the Mean Square Error got even smaller with a value of 0.00278. This result is around the lower end of observed errors across all the methods, and substantially better than for test one and two. However this does not necessarily mean that Linear Regression is the best method for predicting power generated by wind parks by using historical wind parks data from the local and nearby parks. This is because when running the generation for other locations, the predictions tended to give somewhat higher error values. They were still generally lower than observed in test two.

The results from these tests show that Linear Regression can service as an initial approximation of how well the various methods can predict power generation.

8.2.2 K Nearest Neighbours Regression

In the first test, where the goal was to predict power generation using only wind speed, K Nearest Neighbours Regression does a far better job at predicting wind park power generation than the Linear Regression. The Mean Square Error was found to be 0.00417, which is substantially better than the Linear Regression on the same data, and can function as a decent prediction.

The second test, where the goal was to predict power generation using historic wind power generation data, the K Nearest Neighbours algorithms did slightly better with a Mean Square Error of 0.00361. This result is pretty similar to the Linear Regression for test two, which had a Mean Square Error of 0.00369. So K Nearest Neighbours offers little advantage over Linear Regression for this test.

The final test, where the goal was to predict power generation using historic wind power generation data based on the target wind park, and its neighbours the K, Nearest Neighbours achieved a Mean Square Error of 0.0039. This number is higher than when less data was included in the regression, which suggests we are including data that is irrelevant for the prediction, and their inclusion is degrading the prediction quality.

K Nearest Neighbours offers a decent way of generating predictions, but falls short of the methods from Machine Learning which is discussed below in 8.2.3 and 8.2.4.

8.2.3 Support Vector Regression

In the first test, where the goal was to predict power generation using wind speed, Support Vector Regression does a better job than Linear Regression, but slightly worse than K Nearest Neighbours. The results are still not too bad with a Mean Square Error 0.00499.

The second test, where the goal was to predict power generation using historic wind power generation data, the Support Vector Regression generates a prediction that is on par with both Linear Regression and K Nearest neighbours. The Mean Square Error was found to be 0.00351.

In the final test, where the goal was to predict power generation using historical wind power generation data based on the target wind park, and its neighbours, the Support Vector Regression achieved a Mean Square Error of 0.00262. This result is the best prediction found so far in the thesis.

Since the amount of outliers changes from wind park to wind park special tuning of the variables per park may be needed. While an epsilon value of 0.01 was used for park 4155 this may not be the best for a different park.

Particularly on the third test the Support Vector Regression does better than all the previously discussed methods, because of its increased ability to deal with outliers relative to Linear Regression and K Nearest Neighbours.

8.2.4 Multi-Level Perceptron Regression

In the first test, where the goal was to predict power generation using wind speed, Multi-Level Perceptron Regression does a passable job of generating predictions with a Mean Square Error of 0.00475.

In the second test, where the goal was to predict power generation using historic wind power generation data, the Multi-Layer Perceptron regression does a comparable job to the previously discussed methods with a Mean Square Error of 0.00345.

In the final test, where the goal was to predict power generation using historical wind power generation data based on the target wind park, and its neighbours the Multi-Layer Perceptron managed to score the lowest error value seen so far with a Mean Square Error of 0.002544. This was also the best prediction among the tested algorithms found in the thesis.

Like the Support Vector Regression the Multi-Layer Perceptron may need some tuning per park. For some wind parks it may be necessary to use a higher number of neurons to generate good predictions, while for other more predictable parks time can be saved by reducing the number of neurons, and by that decrease the time needed to generate a prediction.

The Multi-Layer Perceptron does not generate the exact same results every time the algorithm is run. The algorithm is internally initialized with random numbers, from which the numbers that happen to align well are emphasized until a prediction can be made. Each time the algorithm is run it will tend to find a new combination of internal descriptions of the data unlike those of the previous runs. These various runs will often not have time to converge, or get stuck with descriptions for which it cannot find a direct improvement from. This last problem is called finding a local optimum.

The Multi-Layer Perceptron regression offers a good way to generate predictions for wind parks, and are better than the other methods at sorting out irrelevant data.

8.3 Sources of Errors

There are several ways in which errors can have affected the results found in this thesis. The first, and possibly most important, is the possible bias in the selection of data. The selection of wind park 4155 is done based on the park appearing to the writer as being representative of a large number of other wind parks. This was based on the idea that this would be one of the harder parks to predict, due to the volatile wind speeds at that location, and that the wind speeds in the park at times surpassed the wind speeds needed to cause the turbines in the wind park to shut down. The winds can be seen in the graphs to pass 33 meters per second, while the turbines cut out at 25 meters per second[7].

By the very nature of predictions looking at past data to predict future outcomes it is also hard to deal with extreme situations that have not been seen in the input data. This could lead to predictions that are very misleading in such cases. In particular we do not know traits the Multi-Layer Perceptron is looking at when generating its results, and it is possible they cause weird results when presented with input not present in their training data.

8.4 Final Thoughts

The Machine Learning algorithms showed promise for getting good predictions, but they are computationally expensive. And truly great results may require a more efficient implementation of the algorithms than used in this thesis.

9 Conclusion

The goal of this thesis was to conclude upon the following question: Can the methods from Machine Learning improve our ability to predict energy generation from wind parks over older methods?

In the case of which input data was relevant for the various methods, the thesis found that some amounts of historic data, as well as some amount of wind power data from neighbouring wind parks was enough to generate good results. However simply supplying more data further back in time, or from wind parks further away did not necessarily improve the predictions. The thesis did not find that including 90 minutes of historic data was better than including only 30. It also found that including wind parks nine kilometers away from the target park was not shown to be better than only including those three kilometers away.

In the case of how well the four methods did when changing the amount of input data. It was found that the Linear Regression could not create prediction when only using Wind Speed data for same-time prediction. However Linear Regression could generate decent results, when it got some more data. K Nearest Neighbours decent predictions throughout, but was surpassed by the Machine Learning algorithms. The Support Vector Regression did better than both Linear Regression and K Nearest Neighbours, particularly as the data-set got larger with the inclusion of both historic data, and data from neighbouring wind parks. Finally the Multi-Layer Perceptron offers a pretty good way of prediction the power generation data for the wind parks, and generates prediction with lower error value than the other methods presented.

To answer the thesis question, this thesis concludes that in the case for predicting the power generation in a single wind park, the Machine Learning algorithms did a good job at generating predictions. As shown in the thesis in all but one illustrated case where Linear Regression generated a great prediction, the Machine Learning algorithms could generate better results than Linear Regression and K Nearest Neighbours. The non-Machine Learning algorithms Linear Regression and K Nearest Neighbours are not generally the best methods for creating wind power generation predictions, but they do generate decent predictions. However these predictions were generally surpassed by the Machine Learning algorithms used in this thesis.

With proper tuning of the input variables, which is possibly location dependent, the Machine Learning algorithm Multi-level Perceptron could generate better predictions than Linear Regression and K Nearest Neighbours, because of its ability to identify which parts of the input data is the most predictive.

9.1 Future Research

While writing this thesis the following questions, among others, came to mind, and have been left unexplored.

Would it be possible to set up a Mult-Layer Perceptron Regression that continually updates its data by including data generated by wind parks after they have been set up and are operational? Could this improve the predictions for that park?

How far into the future is it possible to generate good results? To look into this in any meaningful way, a better understanding which error values are acceptable for use would be needed.

References

[1] Stephen Marsland. *Machine Learning: An Algorithmic Perspective, Second Edition*. 2nd. Chapman & Hall/CRC, 2014. ISBN: 1466583282, 9781466583283.

[2] *matplotlib home page*. URL: `https://matplotlib.org/index.html` (visited on 05/25/2017).

[3] *NREL Corporate Fact Sheet*. URL: `https://www.nrel.gov/docs/fy16osti/66385.pdf` (visited on 05/25/2017).

[4] Cameron W. Potter et al. "Creating the Dataset for the Western Wind and Solar Integration Study (U.S.A.)" In: *Wind Engineering* 32.4 (2008), pp. 325–338. DOI: `10.1260/0309-524X.32.4.325`. eprint: `http://dx.doi.org/10.1260/0309-524X.32.4.325`. URL: `http://dx.doi.org/10.1260/0309-524X.32.4.325`.

[5] *scikit-learn home page*. URL: `http://scikit-learn.org/stable/index.html` (visited on 05/25/2017).

[6] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (Jan. 2016). Article, pp. 484–489. ISSN: 0028-0836. URL: `http://dx.doi.org/10.1038/nature16961`.

[7] *Vestas V90-3.0 Official Information Broshure*. URL: `https://www.vestas.com/en/system/links/media-links/product-media/brochures/uk/v90-30-brochure` (visited on 05/25/2017).

[8] *Western Wind and Solar Integration Study*. URL: `http://www.nrel.gov/docs/fy10osti/47434.pdf` (visited on 05/25/2017).

[9] windML. *windML home page*. URL: `http://www.windml.org/` (visited on 05/23/2017).

[10] Renewable Energy World. *World's Largest Wind Project is Underway*. 2010. URL: `http://www.renewableenergyworld.com/articles/2010/07/worlds-largest-wind-project-is-underway.html` (visited on 05/23/2017).

Appendices

A Code

A.1 LRSpeedPower.py - LR prediction of power using speed

```
import math
import sys
import matplotlib.pyplot as plt
import numpy as np
from windml.datasets.nrel import NREL
from windml.mapping.power_mapping import PowerMapping
from windml.mapping.speed_mapping import SpeedMapping

from sklearn import linear_model
from sklearn.metrics import mean_squared_error


def build_speed_power_plot(speed, power, name, *functions
    ):
    for function in functions:
        plt.plot(speed.reshape(-1,1), function.predict(
            speed.reshape(-1,1)), color='0.05', lw=3.0)

    plt.scatter(speed.reshape(-1,1), power, color='0.25',
        lw =0.0)
    plt.xlabel("Wind Speed")
    plt.ylabel("Wind Power")
    plt.xlim([0, 40])
    plt.ylim([0, 1])
    plt.savefig(name, bbox_inches='tight')

def remove_outliers(x, y):
    to_remove = []
    for i in range(0,len(x)):
        if x[i] > 25:
            to_remove.insert(0,i)
        if x[i] > 10 and y[i] == 0:
            to_remove.insert(0,i)
    x = np.delete(x, to_remove)
    y = np.delete(y, to_remove)
    return (x,y)

def remove_turbine_off(x,y):
    to_remove = []
    for i in range(0,len(x)):
        if y[i] == 0:
            to_remove.insert(0,i)
    x = np.delete(x, to_remove)
    y = np.delete(y, to_remove)
    return (x,y)


def retrieve_speed_power_data(park_id):
    #get turbine at the center of the wind park
    turbine = NREL().get_turbine(park_id, 2004, 2006)

    S_Mapping = SpeedMapping()
    P_Mapping = PowerMapping()

    speed = S_Mapping.get_labels_turbine(turbine, 1, 1)
    power = P_Mapping.get_labels_turbine(turbine, 1, 1)

    return (speed, power)

def split_data(data, ratio):
    data_len = len(data)
    split_pos= int(data_len * ratio)
    out_1 = data[:split_pos]
    out_2 = data[split_pos:]
    return (out_1, out_2)

def sort_data_pairs(x,y):
    shuffle = x.argsort()
    x = x[shuffle]
    y = y[shuffle]
    return (x,y)
if len(sys.argv) != 2:
    print "Give a wind park ID as argument"
    sys.exit()

print "This script generates a plot using Linear Regression"
print "It uses wind speed to predict power generation"

park_id = 4155
if sys.argv[1].isdigit():
    print "Wind park used as argument: " + sys.argv[1]
    park_id =int(sys.argv[1])
else:
        print "No wind park number supplied. Using default 4155"

speed, power = retrieve_speed_power_data(park_id)


#Normalization
power = power/30

#reduction and trimming of data. Optional
#speed, power  = speed[:10000], power[:10000]
#speed, power  = remove_outliers(speed, power)
#speed, power  = remove_turbine_off(speed, power)

#splitting into training and testing data
speed_train, speed_test = split_data(speed, 0.8)
power_train, power_test = split_data(power, 0.8)

#sorting datapoints from low to high wind speed. To make order of events irrellevant
speed_train, power_train = sort_data_pairs(speed_train, power_train)
speed_test , power_test  = sort_data_pairs(speed_test , power_test )

LR = linear_model.LinearRegression().fit(speed_train.reshape(-1,1), power_train)
error = mean_squared_error(power_test, LR.predict(speed_test.reshape(-1,1)))
print "Error Size\n" + str(error)
name = "LRSpeedPowerPark" + str(park_id) + "error" + str(error) + ".png"
build_speed_power_plot(speed_test, power_test, name, LR)
```

A.2 KNNSpeedPower.py - KNN prediction of power using speed

```python
import math
import sys
import matplotlib.pyplot as plt
import numpy as np
from windml.datasets.nrel import NREL
from windml.mapping.power_mapping import PowerMapping
from windml.mapping.speed_mapping import SpeedMapping

from sklearn import linear_model
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error


def build_speed_power_plot(speed, power, name, *functions):
    for function in functions:
        plt.plot(speed.reshape(-1,1), function.predict(speed.reshape(-1,1)), color='0.05', lw=1.0)

    plt.scatter(speed.reshape(-1,1), power, color='0.25', lw =0.0)
    plt.xlabel("Wind␣Speed")
    plt.ylabel("Wind␣Power")
    plt.xlim([0, 40])
    plt.ylim([0, 1])
    plt.savefig(name, bbox_inches='tight')

def remove_outliers(x, y):
    to_remove = []
    for i in range(0,len(x)):
        if x[i] > 25:
            to_remove.insert(0,i)
        if x[i] > 10 and y[i] == 0:
            to_remove.insert(0,i)
    x = np.delete(x, to_remove)
    y = np.delete(y, to_remove)
    return (x,y)

def remove_turbine_off(x,y):
    to_remove = []
    for i in range(0,len(x)):
        if y[i] == 0:
            to_remove.insert(0,i)
    x = np.delete(x, to_remove)
    y = np.delete(y, to_remove)
    return (x,y)


def retrieve_speed_power_data(park_id):
    #get turbine at the center of the wind park
    turbine = NREL().get_turbine(park_id, 2004, 2006)

    S_Mapping = SpeedMapping()
    P_Mapping = PowerMapping()

    speed = S_Mapping.get_labels_turbine(turbine, 1, 1)
    power = P_Mapping.get_labels_turbine(turbine, 1, 1)

    return (speed, power)

def split_data(data, ratio):
    data_len = len(data)
    split_pos= int(data_len * ratio)
    out_1 = data[:split_pos]
    out_2 = data[split_pos:]
    return (out_1, out_2)

def sort_data_pairs(x,y):
    shuffle = x.argsort()
    x = x[shuffle]
    y = y[shuffle]
    return (x,y)

if len(sys.argv) != 3:
    print "Give a wind park ID and a number for K as argument"
    sys.exit()

print "This script generates a plot using K Nearest Neighbours"
print "It uses wind speed to predict power generation"

park_id = 4155
if sys.argv[1].isdigit():
    print "Wind park used as argument: " + sys.argv[1]
    park_id =int(sys.argv[1])
else:
    print "No wind park number supplied. Using default 
```

```
        4155"

speed, power = retrieve_speed_power_data(park_id)

neighbours = 10
if sys.argv[2].isdigit():
    print "number of neighbours used: " + sys.argv[2]
    neighbours = int(sys.argv[2])
else:
    print "No number of neighbours supplied. Using 
        default 10"


#Normalization
power = power/30

#reduction and trimming of data. Optional
#speed, power  = speed[:10000], power[:10000]
#speed, power  = remove_outliers(speed, power)
#speed, power  = remove_turbine_off(speed, power)

#splitting into training and testing data
speed_train, speed_test = split_data(speed, 0.8)
power_train, power_test = split_data(power, 0.8)

#sorting datapoints from low to high wind speed. To make
    order of events irrellevant
speed_train, power_train = sort_data_pairs(speed_train,
    power_train)
speed_test , power_test  = sort_data_pairs(speed_test ,
    power_test )

KNN = KNeighborsRegressor(neighbours, 'uniform').fit(
    speed_train.reshape(-1,1), power_train)
error = mean_squared_error(power_test, KNN.predict(
    speed_test.reshape(-1,1)))
print "Mean Squared Error\n" + str(error)
name = "KNNpeedPowerPark" + str(park_id) + "Neighbours" +
     str(neighbours) + "error" + str(error) + ".png"
build_speed_power_plot(speed_test, power_test, name, KNN)
```

A.3 SVRSpeedPower.py - SVR prediction of power using speed

```
import math
import sys
import matplotlib.pyplot as plt
import numpy as np
from windml.datasets.nrel import NREL
from windml.mapping.power_mapping import PowerMapping
from windml.mapping.speed_mapping import SpeedMapping

from sklearn.metrics import mean_squared_error
from sklearn.svm import SVR

def build_speed_power_plot(speed, power, name, *functions
    ):
    for function in functions:
        plt.plot(speed.reshape(-1,1), function.predict(
            speed.reshape(-1,1)), color='0.05', lw=1.0)

    plt.scatter(speed.reshape(-1,1), power, color='0.25',
        lw =0.0)
    plt.xlabel("Wind Speed")
    plt.ylabel("Wind Power")
    plt.xlim([0, 40])
    plt.ylim([0, 1])
    plt.savefig(name, bbox_inches='tight')

def remove_outliers(x, y):
    to_remove = []
    for i in range(0,len(x)):
        if x[i] > 25:
            to_remove.insert(0,i)
        if x[i] > 10 and y[i] == 0:
            to_remove.insert(0,i)
    x = np.delete(x, to_remove)
    y = np.delete(y, to_remove)
    return (x,y)

def remove_turbine_off(x,y):
    to_remove = []
    for i in range(0,len(x)):
        if y[i] == 0:
            to_remove.insert(0,i)
    x = np.delete(x, to_remove)
    y = np.delete(y, to_remove)
    return (x,y)


def retrieve_speed_power_data(park_id):
    #get turbine at the center of the wind park
    turbine = NREL().get_turbine(park_id, 2004, 2006)

    S_Mapping = SpeedMapping()
    P_Mapping = PowerMapping()

    speed = S_Mapping.get_labels_turbine(turbine, 1, 1)
    power = P_Mapping.get_labels_turbine(turbine, 1, 1)

    return (speed, power)

def split_data(data, ratio):
    data_len = len(data)
    split_pos= int(data_len * ratio)
    out_1 = data[:split_pos]
    out_2 = data[split_pos:]
    return (out_1, out_2)

def sort_data_pairs(x,y):
    shuffle = x.argsort()
    x = x[shuffle]
    y = y[shuffle]
    return (x,y)

if len(sys.argv) == 0:
    print str(sys.argv[0]) + "␣<park_ID>␣<Training_info>␣<gamma>␣<c>"
    print "Give␣a␣wind␣park␣ID␣and␣training␣info␣as␣arguments"
    print "the␣training␣info␣adjusts␣the␣size␣of␣the␣training␣data"
    print "1␣−>␣all␣the␣data.␣2␣−>␣half␣the␣data.␣3␣−>␣a␣third,␣and␣so␣on"
    sys.exit()

print "This␣script␣generates␣a␣plot␣using␣Support␣Vector␣Regression"
print "It␣uses␣wind␣speed␣to␣predict␣power␣generation"
park_id = 4155
if sys.argv[1].isdigit():
    print "Wind␣park␣used␣as␣argument:␣" + sys.argv[1]
```

```python
    park_id =int(sys.argv[1])
else:
    print "No wind park number supplied. Using default 4155"

speed, power = retrieve_speed_power_data(park_id)

training_reduction = 10
if len(sys.argv) > 2:
    print "Training Reduction used as argument: " + sys.argv[2]
    training_reduction = int(sys.argv[2])
else:
    print "No training reduction supplied, using 10"

epsilon = 0.01
if len(sys.argv) > 3:
    print "epsilon used as argument: " +sys.argv[3]
    epsilon = float(sys.argv[3])
else:
    print "No epsilon supplied, using 0.01"

c = 1
if len(sys.argv) > 4:
    print "c used as argument: " + sys.argv[4]
    c = float(sys.argv[4])
else:
    print "No C supplied, using 1"

#Normalization
power = power/30

#reduction and trimming of data. Optional
speed, power  = speed[:len(speed):training_reduction], power[:len(speed):training_reduction]
#speed, power  = remove_outliers(speed, power)
#speed, power  = remove_turbine_off(speed, power)

#splitting into training and testing data
speed_train, speed_test = split_data(speed, 0.8)
power_train, power_test = split_data(power, 0.8)

#sorting datapoints from low to high wind speed. To make order of events irrellevant
speed_train, power_train = sort_data_pairs(speed_train, power_train)
```

```
speed_test , power_test = sort_data_pairs(speed_test ,
    power_test )

SVR = SVR(epsilon=epsilon, C=c).fit(speed_train.reshape
    (-1,1), power_train)
error = mean_squared_error(power_test, SVR.predict(
    speed_test.reshape(-1,1)))
print "Mean Squared Error\n" + str(error)
name = "SVRSpeedPowerPark" + str(park_id) + "
    TrainingReduction" + str(training_reduction) + "
    Epsilon" + str(epsilon) + "c" + str(c) + "error" + str
    (error) + ".png"
build_speed_power_plot(speed_test, power_test, name, SVR)
```

A.4 MLPRSpeedPower.py - MLPR prediction of power using speed

```python
import math
import sys
import matplotlib.pyplot as plt
import numpy as np
from windml.datasets.nrel import NREL
from windml.mapping.power_mapping import PowerMapping
from windml.mapping.speed_mapping import SpeedMapping

from sklearn.metrics import mean_squared_error
from sklearn.neural_network import MLPRegressor

def build_speed_power_plot(speed, power, name, *functions):
    for function in functions:
        plt.plot(speed.reshape(-1,1), function.predict(speed.reshape(-1,1)), color='0.05', lw=1.0)

    plt.scatter(speed.reshape(-1,1), power, color='0.25', lw =0.0)
    plt.xlabel("Wind_Speed")
    plt.ylabel("Wind_Power")
    plt.xlim([0, 40])
    plt.ylim([0, 1])
    plt.savefig(name, bbox_inches='tight')

def remove_outliers(x, y):
    to_remove = []
    for i in range(0,len(x)):
        if x[i] > 25:
            to_remove.insert(0,i)
        if x[i] > 10 and y[i] == 0:
            to_remove.insert(0,i)
    x = np.delete(x, to_remove)
    y = np.delete(y, to_remove)
    return (x,y)

def remove_turbine_off(x,y):
    to_remove = []
    for i in range(0,len(x)):
        if y[i] == 0:
            to_remove.insert(0,i)
    x = np.delete(x, to_remove)
    y = np.delete(y, to_remove)
    return (x,y)


def retrieve_speed_power_data(park_id):
    #get turbine at the center of the wind park
    turbine = NREL().get_turbine(park_id, 2004, 2006)

    S_Mapping = SpeedMapping()
    P_Mapping = PowerMapping()

    speed = S_Mapping.get_labels_turbine(turbine, 1, 1)
    power = P_Mapping.get_labels_turbine(turbine, 1, 1)

    return (speed, power)

def split_data(data, ratio):
    data_len = len(data)
    split_pos= int(data_len * ratio)
    out_1 = data[:split_pos]
    out_2 = data[split_pos:]
    return (out_1, out_2)

def sort_data_pairs(x,y):
    shuffle = x.argsort()
    x = x[shuffle]
    y = y[shuffle]
    return (x,y)

if len(sys.argv) == 0:
    print str(sys.argv[0]) + " <park_ID> <Training_info> 
        <gamma> <c>"
    print "Give a wind park ID and training info as 
        arguments"
    print "the training info adjusts the size of the 
        training data"
    print "1 -> all the data. 2 -> half the data. 3 -> a 
        third, and so on"
    sys.exit()

print "This script generates a plot using A multilayer 
    Perceptron"
print "It uses wind speed to predict power generation"
park_id = 4155
if sys.argv[1].isdigit():
    print "Wind park used as argument: " + sys.argv[1]
    park_id =int(sys.argv[1])
else:
    print "No wind park number supplied. Using default 4155"
speed, power = retrieve_speed_power_data(park_id)

training_reduction = 1
if len(sys.argv) > 2:
    print "Training Reduction used as argument: " + sys.argv[2]
    training_reduction = int(sys.argv[2])
else:
    print "No training reduction supplied, using 1"

layers = 300
if len(sys.argv) > 3:
    print "Layers used as argument: " +sys.argv[3]
    layers = int(sys.argv[3])
else:
    print "No layer number supplied supplied, using " + str(layers)


#Normalization
power = power/30

#reduction and trimming of data. Optional
speed, power  = speed[:len(speed):training_reduction], power[:len(speed):training_reduction]
#speed, power  = remove_outliers(speed, power)
#speed, power  = remove_turbine_off(speed, power)

#splitting into training and testing data
speed_train, speed_test = split_data(speed, 0.8)
power_train, power_test = split_data(power, 0.8)

#sorting datapoints from low to high wind speed. To make order of events irrellevant
speed_train, power_train = sort_data_pairs(speed_train, power_train)
speed_test , power_test  = sort_data_pairs(speed_test , power_test )

MLPR = MLPRegressor(hidden_layer_sizes=layers).fit(speed_train.reshape(-1,1), power_train)
error = mean_squared_error(power_test, MLPR.predict(
    speed_test.reshape(-1,1)))
print "Mean Squared Error\n" + str(error)
name = "MLPRSpeedPowerPark" + str(park_id) + "
    TrainingReduction" + str(training_reduction) + "
    LayerSize" + str(layers) + "error" + str(error) + ".
    png"
build_speed_power_plot(speed_test, power_test, name, MLPR
    )
```

A.5 LRPrediction.py - LR prediction of power using same or nearby park power

```
import math
import sys
import matplotlib.pyplot as plt
import numpy as np
from windml.datasets.nrel import NREL
from windml.mapping.power_mapping import PowerMapping
from windml.mapping.speed_mapping import SpeedMapping

from sklearn import linear_model
from sklearn.metrics import mean_squared_error

#splits a numpy array in two to a set ratio ex. 0.8
def split_data(data, ratio):
    data_len = len(data)
    split_pos= int(data_len * ratio)
    out_1 = data[:split_pos]
    out_2 = data[split_pos:]
    return (out_1, out_2)


print "LRPrediction.py <park_id> <Training_reduction> <wind_park_radius> <prediction_distance> <window_size>"

park_id = 4155
if len(sys.argv) > 1:
    print "Wind park used as argument: " + sys.argv[1]
    park_id =int(sys.argv[1])
else:
    print "No wind park number supplied. Using default 4155"

training_reduction = 1
if len(sys.argv) > 2:
    print "Training Reduction used as argument: " + sys.argv[2]
    training_reduction = int(sys.argv[2])
else:
    print "No training reduction supplied, using 1"

wind_park_radius = 1
if len(sys.argv) > 3:
    print "Wind park Radius used as argument: " + sys.
```

```
        argv[3]
    wind_park_radius = int(sys.argv[3])
else:
    print "No␣wind␣park␣radius␣supplied,␣using␣1"

prediction_distance = 1
if len(sys.argv) > 4:
    print "Prediction␣distance␣used␣as␣argument:␣" + sys.argv[4]
    prediction_distance = int(sys.argv[4])
else:
    print "No␣prediction␣distance␣supplied,␣using␣1"

window_size = 3
if len(sys.argv) > 5:
    print "window_size␣used␣as␣argument:␣" + sys.argv[5]
    window_size = int(sys.argv[5])
else:
    print "No␣window_size␣supplied,␣using␣3"



#define a windpark using a turbine ID. techapi = 4155
#size in km and dates as arguments
windpark = NREL().get_windpark(park_id,
                               wind_park_radius,
                               2004,
                               2006)

#the turbine in the exact center of the windpark is the one we're trying to predict
target_turbine = windpark.get_target()

p_mapping = PowerMapping()

print "number␣of␣10␣minute␣timesteps␣into␣the␣future:␣" + str(prediction_distance)

#retrieve power generated by turbines in park
turbine_power_matrix = p_mapping.get_features_park(windpark, window_size, prediction_distance)
target_values = p_mapping.get_labels_turbine(target_turbine, window_size, prediction_distance)

#normalize to 0-1
turbine_power_matrix = turbine_power_matrix/30
```

```
target_values = target_values/30


#reduction in data size using training_reduction 1 is all
    , 2 is half and so on
turbine_power_matrix = turbine_power_matrix[:len(
    turbine_power_matrix):training_reduction]
target_values = target_values[:len(target_values):
    training_reduction]

turbine_power_matrix_train, turbine_power_matrix_test =
    split_data(turbine_power_matrix, 0.8)
target_values_train, target_values_test = split_data(
    target_values, 0.8)

#train a regressor based on the training data
#Linear Regression
LR = linear_model.LinearRegression().fit(
    turbine_power_matrix_train, target_values_train)
prediction = LR.predict(turbine_power_matrix_test)
error = mean_squared_error(target_values_test, prediction
    )
print "MSE LR " + str(error)

time = np.array(range(0, len(turbine_power_matrix_test)))
plt.plot(time, target_values_test, label="observed values
    ", color='0.0', lw=1.5)
plt.plot(time, prediction, label="predicted values LR",
    color='0.0', lw=1.5, ls='--')
plt.xlabel("Time")
plt.ylabel("Wind Power")
plt.ylim([0, 1])
plt.xlim([1000, 1100])
plt.legend()
#plt.show()

name = "LRPredictionPark" + str(park_id) + "
    TrainingReduction" + str(training_reduction) + "
    ParkRadius" + str(wind_park_radius) + "
    PredictionDistance" + str(prediction_distance) + "
    WindowSize" + str(window_size) + "Error" + str(error)
    + ".png"

plt.savefig(name, bbox_inches='tight')
```

A.6 KNNPrediction.py - KNN prediction of power using same or nearby park power

```python
import math
import sys
import matplotlib.pyplot as plt
import numpy as np
from windml.datasets.nrel import NREL
from windml.mapping.power_mapping import PowerMapping
from windml.mapping.speed_mapping import SpeedMapping

from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

#splits a numpy array in two to a set ratio ex. 0.8
def split_data(data, ratio):
    data_len = len(data)
    split_pos= int(data_len * ratio)
    out_1 = data[:split_pos]
    out_2 = data[split_pos:]
    return (out_1, out_2)


print "LRPrediction.py <park_id> <Training_reduction> <wind_park_radius> <prediction_distance> <window_size> <Size for K>"

park_id = 4155
if len(sys.argv) > 1:
    print "Wind park used as argument: " + sys.argv[1]
    park_id =int(sys.argv[1])
else:
    print "No wind park number supplied. Using default 4155"

training_reduction = 1
if len(sys.argv) > 2:
    print "Training Reduction used as argument: " + sys.argv[2]
    training_reduction = int(sys.argv[2])
else:
    print "No training reduction supplied, using 1"

wind_park_radius = 1
if len(sys.argv) > 3:
    print "Wind park Radius used as argument: " + sys.argv[3]
    wind_park_radius = int(sys.argv[3])
else:
    print "No wind park radius supplied, using 1"

prediction_distance = 1
if len(sys.argv) > 4:
    print "Prediction distance used as argument: " + sys.argv[4]
    prediction_distance = int(sys.argv[4])
else:
    print "No prediction distance supplied, using 1"

window_size = 3
if len(sys.argv) > 5:
    print "window_size used as argument: " + sys.argv[5]
    window_size = int(sys.argv[5])
else:
    print "No window_size supplied, using 3"

K = 25
if len(sys.argv) > 6:
    print "K used as argument: " + sys.argv[6]
    K = int(sys.argv[6])
else:
    print "No K supplied, using " + str(K)

#define a windpark using a turbine ID. techapi = 4155
#size in km and dates as arguments
windpark = NREL().get_windpark(park_id,
                               wind_park_radius,
                               2004,
                               2006)

#the turbine in the exact center of the windpark is the one we're trying to predict
target_turbine = windpark.get_target()

p_mapping = PowerMapping()

print "number of 10 minute timesteps into the future: " + str(prediction_distance)

#retrieve power generated by turbines in park
turbine_power_matrix = p_mapping.get_features_park(
    windpark, window_size, prediction_distance)
target_values = p_mapping.get_labels_turbine(
    target_turbine, window_size, prediction_distance)

#normalize to 0-1
turbine_power_matrix = turbine_power_matrix/30
target_values = target_values/30


#reduction in data size using training_reduction 1 is all
    , 2 is half and so on
turbine_power_matrix = turbine_power_matrix[:len(
    turbine_power_matrix):training_reduction]
target_values = target_values[:len(target_values):
    training_reduction]

turbine_power_matrix_train, turbine_power_matrix_test =
    split_data(turbine_power_matrix, 0.8)
target_values_train, target_values_test = split_data(
    target_values, 0.8)

#train a regressor based on the training data
#Linear Regression
KNN = KNeighborsRegressor(K).fit(
    turbine_power_matrix_train, target_values_train)
prediction = KNN.predict(turbine_power_matrix_test)
error = mean_squared_error(target_values_test, prediction
    )
print "MSE LR " + str(error)

time = np.array(range(0, len(turbine_power_matrix_test)))
plt.plot(time, target_values_test, label="observed values
    ", color='0.0', lw=1.5)
plt.plot(time, prediction, label="predicted values KNN",
    color='0.0', lw=1.5, ls='--')
plt.xlabel("Time")
plt.ylabel("Wind Power")
plt.ylim([0, 1])
plt.xlim([3750, 3950])
plt.legend()
#plt.show()

name = "KNNPredictionPark" + str(park_id) + "
    TrainingReduction" + str(training_reduction) + "
    ParkRadius" + str(wind_park_radius) + "
    PredictionDistance" + str(prediction_distance) + "
    WindowSize" + str(window_size) + "K" + str(K) + "Error
    " + str(error) + ".png"

plt.savefig(name, bbox_inches='tight')
```

A.7 SVRPrediction.py - SVR prediction of power using same or nearby park power

```
import math
import sys
import matplotlib.pyplot as plt
import numpy as np
from windml.datasets.nrel import NREL
from windml.mapping.power_mapping import PowerMapping
from windml.mapping.speed_mapping import SpeedMapping

from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

#splits a numpy array in two to a set ratio ex. 0.8
def split_data(data, ratio):
    data_len = len(data)
    split_pos= int(data_len * ratio)
    out_1 = data[:split_pos]
    out_2 = data[split_pos:]
    return (out_1, out_2)


print "SVRPrediction.py <park_id> <Training_reduction> <
    wind_park_radius> <prediction_distance> <window_size>
    <epsilon_value> <c_value>"

park_id = 4155
if len(sys.argv) > 1:
    print "Wind park used as argument: " + sys.argv[1]
    park_id =int(sys.argv[1])
else:
    print "No wind park number supplied. Using default
        4155"

training_reduction = 5
if len(sys.argv) > 2:
    print "Training Reduction used as argument: " + sys.
        argv[2]
    training_reduction = int(sys.argv[2])
else:
    print "No training reduction supplied, using " + str(
        training_reduction)

wind_park_radius = 1
```

```
if len(sys.argv) > 3:
    print "Wind park Radius used as argument: " + sys.
        argv[3]
    wind_park_radius = int(sys.argv[3])
else:
    print "No wind park radius supplied, using 1"

prediction_distance = 1
if len(sys.argv) > 4:
    print "Prediction distance used as argument: " + sys.
        argv[4]
    prediction_distance = int(sys.argv[4])
else:
    print "No prediction distance supplied, using 1"

window_size = 3
if len(sys.argv) > 5:
    print "window_size used as argument: " + sys.argv[5]
    window_size = int(sys.argv[5])
else:
    print "No window_size supplied, using 3"

epsilon = 0.01
if len(sys.argv) > 6:
    print "epsilon used as argument: " + sys.argv[6]
    epsilon = float(sys.argv[6])
else:
    print "No epsilon supplied, using " + str(epsilon)

c = 1
if len(sys.argv) > 7:
    print "c used as argumetn: " + sys.argv[7]
    c = int(sys.argv[7])
else:
    print "No c supplied, using " + str(c)

#define a windpark using a turbine ID. techapi = 4155
#size in km and dates as arguments
windpark = NREL().get_windpark(park_id,
                                wind_park_radius,
                                2004,
                                2006)

#the turbine in the exact center of the windpark is the
    one we're trying to predict
target_turbine = windpark.get_target()
```

```

p_mapping = PowerMapping()

print "number of 10 minute timesteps into the future: " +
     str(prediction_distance)

#retrieve power generated by turbines in park
turbine_power_matrix = p_mapping.get_features_park(
    windpark, window_size, prediction_distance)
target_values = p_mapping.get_labels_turbine(
    target_turbine, window_size, prediction_distance)

#normalize to 0-1
turbine_power_matrix = turbine_power_matrix/30
target_values = target_values/30


#reduction in data size using training_reduction 1 is all
    , 2 is half and so on
turbine_power_matrix = turbine_power_matrix[:len(
    turbine_power_matrix):training_reduction]
target_values = target_values[:len(target_values):
    training_reduction]

turbine_power_matrix_train, turbine_power_matrix_test =
    split_data(turbine_power_matrix, 0.8)
target_values_train, target_values_test = split_data(
    target_values, 0.8)

#train a regressor based on the training data
SVR = SVR(epsilon=epsilon, C=c).fit(
    turbine_power_matrix_train, target_values_train)

prediction = SVR.predict(turbine_power_matrix_test)
error = mean_squared_error(target_values_test, prediction
    )
print "MSE LR " + str(error)

time = np.array(range(0, len(turbine_power_matrix_test)))
plt.plot(time, target_values_test, label="observed values
    ", color='0.0', lw=1.5)
plt.plot(time, prediction, label="predicted values SVR",
    color='0.0', lw=1.5, ls='--')
plt.xlabel("Time")
plt.ylabel("Wind Power")
plt.ylim([0, 1])
plt.xlim([1300, 1500])
plt.legend()
#plt.show()

name = "SVRPredictionPark" + str(park_id) + "
    TrainingReduction" + str(training_reduction) + "
    ParkRadius" + str(wind_park_radius) + "
    PredictionDistance" + str(prediction_distance) + "
    WindowSize" + str(window_size) + "Epsilon" + str(
    epsilon) + "C" + str(c) + "Error" + str(error) + ".png
    "

plt.savefig(name, bbox_inches='tight')
```

A.8 MLPRPrediction.py - MLPR prediction of power using same or nearby park power

```python
import math
import sys
import matplotlib.pyplot as plt
import numpy as np
from windml.datasets.nrel import NREL
from windml.mapping.power_mapping import PowerMapping
from windml.mapping.speed_mapping import SpeedMapping

from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error

#splits a numpy array in two to a set ratio ex. 0.8
def split_data(data, ratio):
    data_len = len(data)
    split_pos= int(data_len * ratio)
    out_1 = data[:split_pos]
    out_2 = data[split_pos:]
    return (out_1, out_2)


print "MLPRPrediction.py <park_id> <Training_reduction> <
    wind_park_radius> <prediction_distance> <window_size>
    <Hidden layer size> <hidden layer depth>"

park_id = 4155
if len(sys.argv) > 1:
    print "Wind park used as argument: " + sys.argv[1]
    park_id =int(sys.argv[1])
else:
    print "No wind park number supplied. Using default
        4155"

training_reduction = 5
if len(sys.argv) > 2:
    print "Training Reduction used as argument: " + sys.
        argv[2]
    training_reduction = int(sys.argv[2])
else:
    print "No training reduction supplied, using " + str(
        training_reduction)

wind_park_radius = 1
```

```
if len(sys.argv) > 3:
    print "Wind␣park␣Radius␣used␣as␣argument:␣" + sys.argv[3]
    wind_park_radius = int(sys.argv[3])
else:
    print "No␣wind␣park␣radius␣supplied,␣using␣1"

prediction_distance = 1
if len(sys.argv) > 4:
    print "Prediction␣distance␣used␣as␣argument:␣" + sys.argv[4]
    prediction_distance = int(sys.argv[4])
else:
    print "No␣prediction␣distance␣supplied,␣using␣1"

window_size = 3
if len(sys.argv) > 5:
    print "window_size␣used␣as␣argument:␣" + sys.argv[5]
    window_size = int(sys.argv[5])
else:
    print "No␣window_size␣supplied,␣using␣3"

hidden_layer_size = 100
if len(sys.argv) > 6:
    print "hidden␣layer␣size␣used␣as␣argument:␣" + sys.argv[6]
    hidden_layer_size = int(sys.argv[6])
else:
    print "No␣hidden␣layer␣size␣supplied,␣using␣" + str(hidden_layer_size)

hidden_layer_depth = 1
if len(sys.argv) > 7:
    print "hidden␣layer␣depth␣␣used␣as␣argumetn:␣" + sys.argv[7]
    hidden_layer_depth = int(sys.argv[7])
else:
    print "No␣hidden␣layer␣depth␣supplied,␣using␣" + str(hidden_layer_depth)

#define a windpark using a turbine ID. techapi = 4155
#size in km and dates as arguments
windpark = NREL().get_windpark(park_id,
                               wind_park_radius,
                               2004,
                               2006)
```

```python

#the turbine in the exact center of the windpark is the one we're trying to predict
target_turbine = windpark.get_target()

p_mapping = PowerMapping()

print "number of 10 minute timesteps into the future: " + str(prediction_distance)

#retrieve power generated by turbines in park
turbine_power_matrix = p_mapping.get_features_park(windpark, window_size, prediction_distance)
target_values = p_mapping.get_labels_turbine(target_turbine, window_size, prediction_distance)

#normalize to 0-1
turbine_power_matrix = turbine_power_matrix/30
target_values = target_values/30


#reduction in data size using training_reduction 1 is all, 2 is half and so on
turbine_power_matrix = turbine_power_matrix[:len(turbine_power_matrix):training_reduction]
target_values = target_values[:len(target_values):training_reduction]

turbine_power_matrix_train, turbine_power_matrix_test = split_data(turbine_power_matrix, 0.8)
target_values_train, target_values_test = split_data(target_values, 0.8)

#create tuple of percreptron hidden layers
hidden_layers_tuple = (hidden_layer_size,)
for i in range(1, hidden_layer_depth):
    hidden_layers_tuple = hidden_layers_tuple + (hidden_layer_size,)

print "Hidden layer layout"
print hidden_layers_tuple

#train a regressor based on the training data
MLPR = MLPRegressor(hidden_layer_sizes=hidden_layers_tuple).fit(turbine_power_matrix_train, target_values_train)
```

```

prediction = MLPR.predict(turbine_power_matrix_test)
error = mean_squared_error(target_values_test, prediction
    )
print "MSE " + str(error)

time = np.array(range(0, len(turbine_power_matrix_test)))
plt.plot(time, target_values_test, label="observed values
    ", color='0.0', lw=1.5)
plt.plot(time, prediction, label="predicted values MLPR",
     color='0.0', lw=1.5, ls='--')
plt.xlabel("Time")
plt.ylabel("Wind Power")
plt.ylim([0, 1])
plt.xlim([5050, 5250])
plt.legend()
#plt.show()

name = "MLPRPredictionPark" + str(park_id) + "
    TrainingReduction" + str(training_reduction) + "
    ParkRadius" + str(wind_park_radius) + "
    PredictionDistance" + str(prediction_distance) + "
    WindowSize" + str(window_size) + "HLSize" + str(
    hidden_layer_size) + "HLDepth" + str(
    hidden_layer_depth) + "Error" + str(error) + ".png"

plt.savefig(name, bbox_inches='tight')
```