jamk.fi

Forecasting Nord Pool's Day-ahead hourly spot prices in Finland

Joni Korpihalkola

Bachelor's thesis
May 2019
Technology, Communication and Transport
Degree Programme in Information and Communication Technology

jamk.fi

Description

Author(s)	Type of publication	Date
Korpihalkola, Joni	Bachelor's thesis	May 2019
		Language of publication: English
	Number of pages: 82	Permission for web publication: x

Title of publication

Forecasting Nord Pool's Day-ahead hourly spot prices in Finland

Degree programme

Information and Communication Technology

Supervisor(s)

Ari Rantala, Esa Salmikangas

Assigned by

Mika Rantonen

Abstract

The liberalization of electricity markets has launched an interest in forecasting future prices and developing models on how the prices will develop. Improvements in computing capabilities have allowed machine learning models and neural networks to be trained faster. The growing amount of open data and access to open APIs improves the performance of neural networks.

The goal of the research was to create a forecasting model that would predict Spot prices in Nord Pool's Day-ahead market in Finland with open-source software. An analysis of Spot price drivers was conducted, and a dataset with multiple different features ranging from weather data to production plans was constructed. Different statistical models generated forecasts from Spot price history and machine learning models were trained on the constructed dataset. The forecasts were compared to a baseline model using three different error metrics.

The result was an ensemble of statistical and machine learning models, where the models' forecasts were combined and given weights by a neural network acting as a metalearner. The model is able to forecast the trend and seasonality of Spot prices but unable to predict sudden price spikes.

The research goal was achieved; however further feature engineering and neural network optimization could improve the model accuracy and produce better forecasts.

Keywords/tags (subjects)

feature engineering, forecasting, neural network, machine learning, Python

Miscellaneous (Confidential information)

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Korpihalkola, Joni	Opinnäytetyö, AMK	Toukokuu 2019
	Sivumäärä 82	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Forecasting Nord Pool's Day-ahead hourly spot prices in Finland

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Ari Rantala, Esa Salmikangas

Toimeksiantaja(t)
Mika Rantonen

Tiivistelmä

Sähkömarkkinoiden avautuminen kilpailulle on synnyttänyt kiinnostusta tulevien hintojen ennustamiseen sekä mallien luomiseen, joiden avulla ennustettaisiin hinnan kehitystä. Tietokoneiden laskentatehon kasvu on mahdollistanut koneoppimisen mallien ja neuroverkkojen kouluttamisen nopeassa ajassa. Avoimen datan lisääntyminen ja API-rajapintojen avautuminen nostaa saatavan datan määrää, mikä parantaa neuroverkkojen suorituskykyä.

Tutkimuksen tavoitteena oli luoda ennustava malli, joka ennustaisi avoimen lähdekoodin ohjelmistoilla Nord Pool -sähköpörssissä Suomen tulevia Spot-hintoja. Spot-hintaan vaikuttavia ominaisuuksia etsittiin data-analyysin metodeilla. Löydettyjen ominaisuuksien avulla luotiin datajoukko, jossa ominaisuudet vaihtelevat säätiedoista sähkön tuotantosuunnitelmiin. Erilaiset tilastolliset mallit loivat ennustuksia Spot-hinnan historian perusteella ja koneoppimisen mallit koulutettiin luodulla datajoukolla. Ennustuksia verrattiin yksinkertaisen mallin ennustuksiin käyttäen kolmea eri virhemittaria.

Tuloksena oli kokoelma tilastollisia ja koneoppimisen malleja, joiden ennustuksille määritettiin painoarvot erillisellä neuroverkolla. Painoarvotetuilla ennustuksilla muodostettiin yhdistetty ennustus. Kokoelma pystyy ennustamaan Spot-hintojen trendin ja kausivaihtelun, mutta mallit eivät osaa ennakoida äkkinäisiä hintapiikkejä.

Tavoite saavutettiin, mutta laajempi ominaisuuksien suunnittelu ja neuroverkon optimisointi nostaisi mallin tarkkuutta ja muodostaisi parempia ennustuksia.

Avainsanat (asiasanat)
ennustaminen, neuroverkot, koneoppiminen, ominaisuuksien suunnittelu, Python

Muut tiedot (Salassa pidettävät liitteet)

Contents

Figures

Tables

Terminology

API	Application Programming Interface
CNN	Convolutional Neural Network
LSTM	Long short-term memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
UTC	Universal Time Coordinated

1 Introduction

1.1 Objective

The objective was to create a model that forecasts day-ahead Finnish area spot prices in Nord Pool's Elspot market. The electricity prices are affected by many variables, such as electricity consumption, outside temperature and electricity production plans. The goal is to find the relevant variables using data analysis methods and create a dataset. The dataset will be used to train machine learning models and neural networks to predict future spot prices. Statistical models will also be used to forecast spot prices based on price history data.

The thesis was made for the Jyväskylä's University of Applied Sciences' project "New Business Innovations from Data analytics", which has received 300 000 euros in funding by the European Regional Development Fund. The goal of the project is to advance knowledge about data analytics and find ways to enrich data for energy sector companies in different pilot cases.

1.2 Nord Pool

Nord Pool is a public company, that is owned by transmission system operators Stattnet, Svenska Kraftnät, Fingrid, Energinet.dk, Elering, Litgrid and Augstprieguma Tikls AS. In 1991, the Norwegian parliament deregulated the electricity market and in 1996, a power exchange by the name of Nord Pool ASA was created between Norway and Sweden. Finland joined the power exchange in 1998 and Denmark followed two years later, which meant that all the Nordic countries were part of the Nord Pool exchange by 2000. (History N.d.)

Nord Pool operates on two different markets, Elspot market and Elbas market. Elspot market is the day-ahead market, which is the focus of this thesis. Elbas allows intra-

day trading and 24/7 electricity trading in the case of sudden changes in consumption or other situations that cannot be covered in the Elspot market. (Voronin 2013, 30.)

1.3 Elspot market

The participants in the Nord Pool Elspot market are mostly electricity retailers and large production plants on the buyer side and owners of large power plants on the seller side. Elspot market is divided into bidding areas, each with a transmission system operator responsible for monitoring congestion in their grid. Physical power contracts are traded in the Elspot market for the next day, which is why it is also referred to as a day-ahead market. Contracts need to have atleast 0.1 MW/h of electricity delivered at minimum. Participants in the Elspot market have until 12:00 CET to submit their purchase and sell orders for the 24-hour period on the next day. (Voronin 2013, 26.)

It is possible for spot market members to submit hourly orders, block orders, exclusive group orders and flexi orders. Hourly orders can be price independent or price dependent. In price independent orders, a buyer can e.g. submit a bid where they are willing to buy 10 MW/h of electricity for the next day, the price for each hour will be determined between the minimum and maximum that the buyer has set. In price dependent orders, a seller could determine that they want to sell 20 MW/h of electricity at 12:00 CET between 40 and 50 euros. If the calculated system price lower than the minimum that the seller set, the volume traded will be lower than 20 MW/h, and the amount to be delivered will be calculated by Nord Pool with linear interpolation. The other types of orders will not be explained further in this thesis, because they are not relevant to the objective. (Day-ahead trading order types N.d.)

After the deadline has passed, the orders are aggregated into two curves: supply and demand. These curves determine the system price. At this point, transmission restrictions are not considered. If there are areas with a surplus of electricity and areas with a deficit, electricity will be transferred from the surplus areas to the deficit areas

with full utilization of transmission capacity as illustrated in Figure 1. (Price Calculation N.d.)

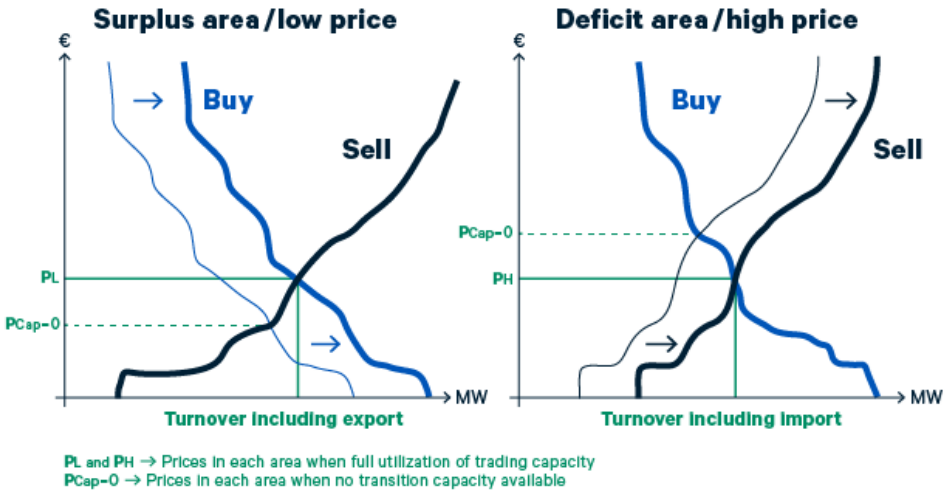

Figure 1 Identical area prices achieved by energy flow from surplus areas to deficit areas.

Due to limited transmission capability in the electricity grid, the system price calculated from aggregated supply and demand curves cannot be applied to all bidding areas. After the system price is calculated, possible bottlenecks caused by transmission are analyzed. If bottlenecks are found, then area prices will be different from the system price and calculated separately. If the electricity grid is not sufficient enough to deliver electricity to an area lacking in electricity, the area price will be higher than the system price in that area. Vice versa, the region overproducing electricity will have an area price that is lower than the system price. These price shifts are performed to encourage customers in low supply areas to purchase energy from local suppliers and in the overproducing areas to incentivize customers to buy cheaper energy. (Voronin 2013, 26-28.)

1.4 Electricity situation in the Nordic region

As can be seen in the graph below in Figure 2, hydro power generates the majority of electricity in the Nordics, with Norway generating almost all their consumed energy with it. Sweden's energy generation is mostly nuclear and hydro power, with a sizable amount of wind power and biomass. (Statistical Factsheet 2017.)

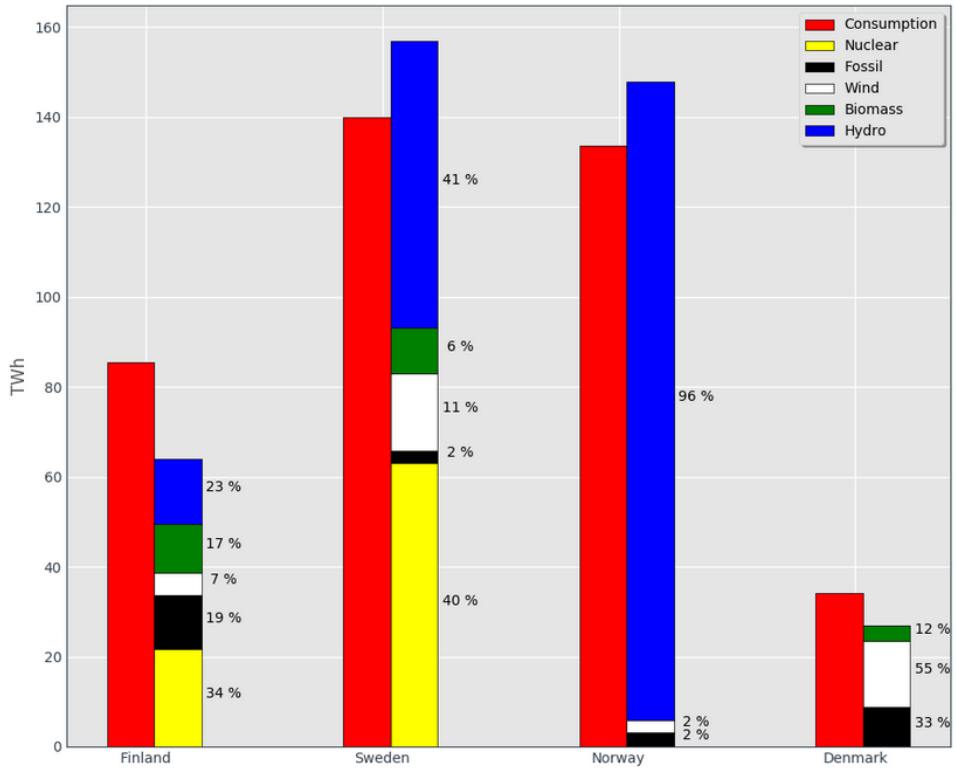

Figure 2. Electricity generation by source and consumption in the Nordic countries

High energy consumption in Finland, Norway and Sweden is due to cold winters, electricity heated houses and heavy industry. Denmark consumes a great deal less electricity than the other Nordic countries, which is due to the lack of heavy industry and milder weather. Energy consumption Finnish industry was 47% of the total consumption as illustrated in Figure 3. Forest industry specifically consumes the most energy, followed by metal industry. (Electricity consumption by Year(*preliminary), Sector and Data. 2018)

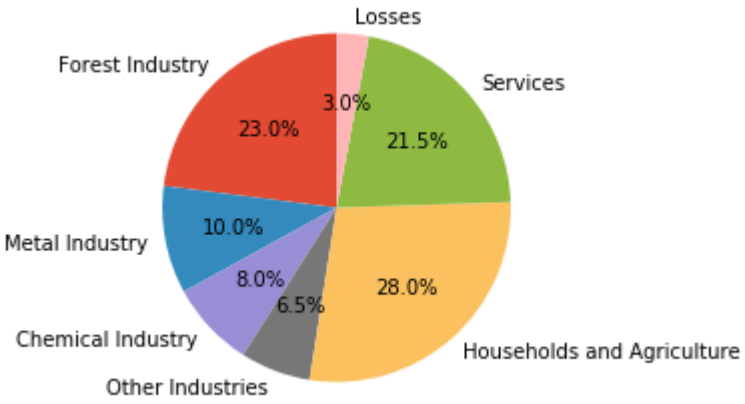

Figure 3. Electricity consumption by sector in Finland in 2018

Housing and agriculture include electricity heating costs; due to long and cold winters it is a major source of energy consumption. In 2017, 22.5 GWh of electricity was used in housing total, with 10.7 GWh was used to heating of spaces as illustrated in Table 1. Household appliances, e.g. televisions and stoves, accounted for 35% of total housing electricity usage. The popularity of saunas is also seen on the table, since heating of saunas is about 5.4% of total housing electricity consumption. (Energy consumption in households 2017.)

Table 1. Energy and electricity consumption in Finnish households by consumption source

Consumption source	**Electricity usage (MWh)**	**Total energy usage (MWh)**
Housing	22,513	66,486
Space heating	10,727	45,349
Household appliances	8,034	8,034

Heating of saunas	1,222	3,057
Heating of water	2,530	9,954

1.5 Energy flows

Sweden and Norway are net exporters of electricity in the Nordic region as seen in Table 2. Most of the energy is exported to Finland, and a sizable amount to Denmark, Lithuania and Poland. Norway exports most of its energy to Sweden and more than 5,000 GWh to Denmark and the Netherlands. Finland generates domestically 76% of consumption; hence imports are needed to sustain heavy industry and electricity housing heating. In 2017 the energy flow balance, which is imports subtracted by exports, was 20,443 GWh. Sweden exports 15,622 GWh of electricity, and Russia is the second largest exporter of energy to Finland with 5,800 GWh, which is slightly more than third the amount of energy imported from Sweden. Finland is a net exporter of electricity to Estonia, with a trade balance of 809 GWh. (Statistical Factsheet 2017.)

Table 2. Energy flows in Nordic countries

Country	**Total energy imported (GWh)**	**Total energy exported (GWh)**	**Energy trade balance (GWh)**
Denmark	15,334	10,620	4,714
Finland	22,590	2,147	20,443
Norway	5,904	20,830	-14,926
Sweden	13,831	32,982	-19,151

1.6 Electricity prices

Electricity prices are highly volatile due to the fact that there is no economically viable way to store large amounts of electricity. Unexpected demands in electricity, shortages, transmission failures, generator failures are some of the usual causes for price spikes. Since the spot price is dictated by a market, some price spikes may occur due to market gaming or false speculations. (Knapik 2017, 1-2.)

Beyond sudden spikes in the price, the cost of electricity in Finland is driven by temperature, power production, congestions in transmission and the water levels of hydro reservoirs in Norway and Sweden. Hydro- and nuclear power production plants are slow and expensive to start; however, they have a low cost while running. These two means of electricity generation form the base electricity generation. When electricity consumption exceeds this base generation, more electricity is generated with combined heat and power facilities, coal and oil condensing or gas turbines. These plants are quick to start production, some even in 30 minutes, but they have a high running cost. The high cost is because of EU emission allowances, the plants must pay for the emissions they cause on top of the normal running cost. (Voronin 2013, 18.)

While electricity itself cannot be stored, hydro reservoir levels act as potential energy that is stored and later used to produce electricity. When the reservoir levels are high, sudden electricity demand spikes can be lessened by hydro plants generating more energy. When the levers are lower, the plant owners may be hesitant to lower their reservoir levels even more to offset demand spikes. This also means that rain forecasts in the Nordics may influence spot prices, since the rains directly affect hydro reservoir levels. (Knapik 2017, 15-16.)

The hot and dry summer of 2018 caused spot prices to rise compared to previous summers. Normally reservoir levels are on a continuous rise after spring, when the

winter snows are melting. The graph in Figure 4 visualizes how water reservoir levels in Norway failed to rise as rapidly in 2018 as they did in 2017, and how daily spot price in Finland also climbed compared to the previous summer. (Leskinen 2018.)

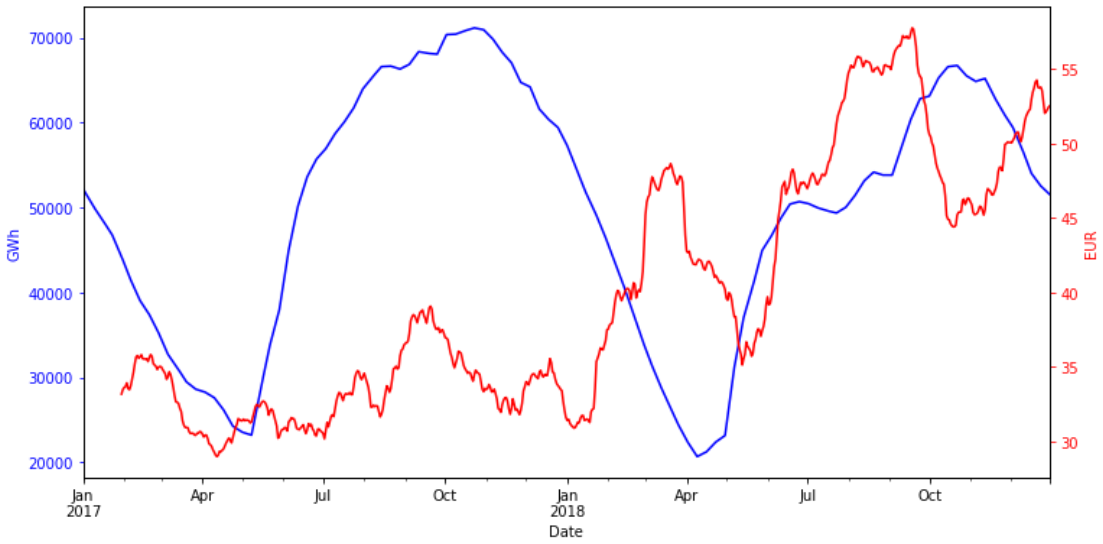

Figure 4. Spot price (red) and hydro reservoir levels in Norway (blue) in 2017-2018

Since the Nordic countries have cold winters, the outside temperature and weather forecasts influence spot prices heavily as illustrated in Figure 5. Sometimes plants producing heat and electricity may lower the spot price due to them producing extra electricity to the grid. (Knapik 2017, 17.)

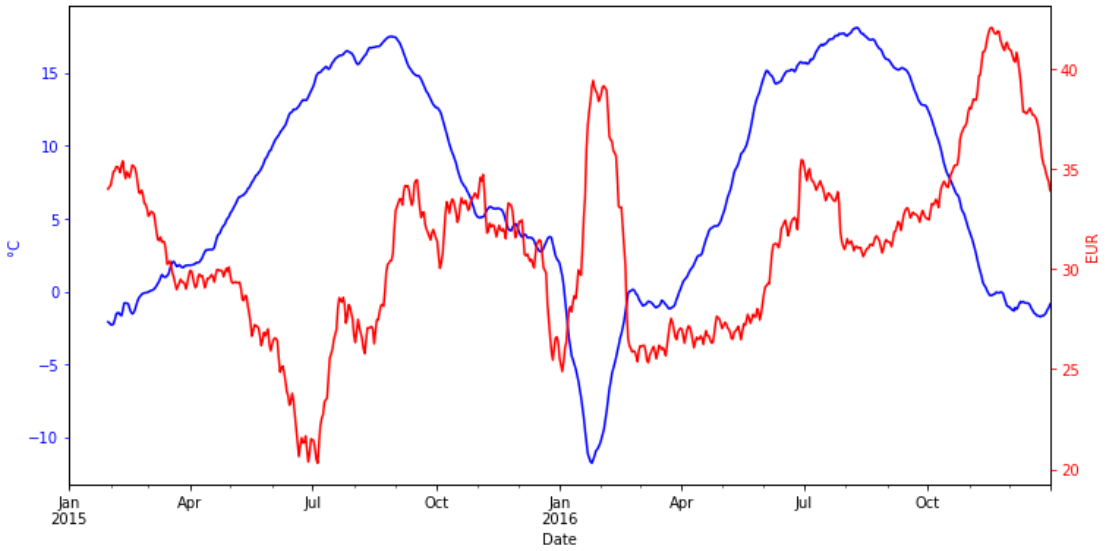

Figure 5. Spot price (red) and temperature in Helsinki (blue) in 2015-2016

Mild summers, on the other hand, decrease consumption, because air conditioning in households is not widespread as seen in Figure 6. The temperature is not inversely proportional to the spot price like the consumption is not proportional to spot price either, which proves the point that the market has other disruptors and price drivers. (Knapik 2017, 17.)

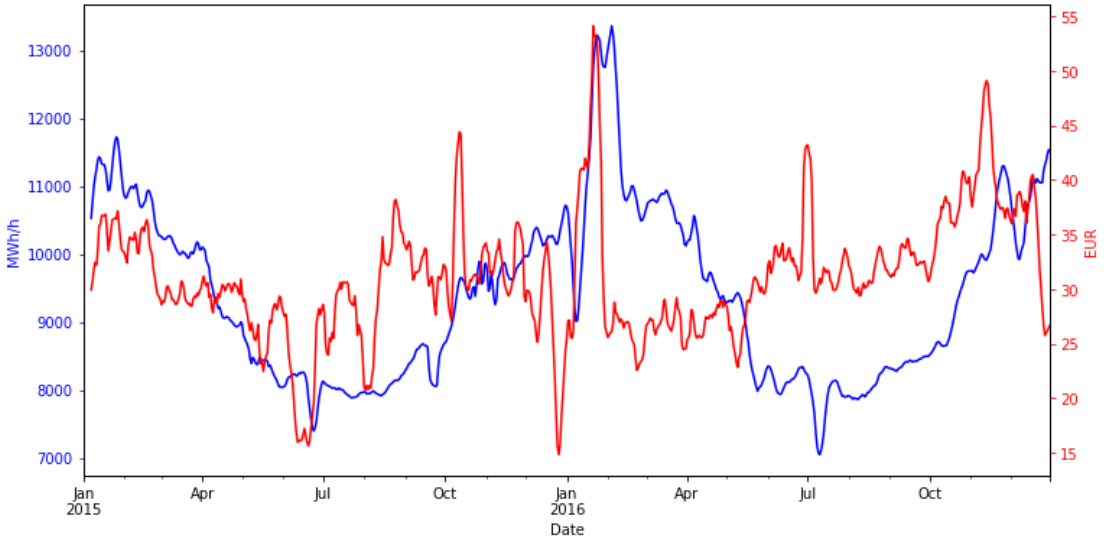

Figure 6. Spot price (red) and consumption (blue) in Finland in 2015-2017

Wind energy has an unpredictability factor to it, where production can either be more than what was forecast or lower. In some cases when wind power production rises, the spot price is cheaper as seen in Figure 7. (Knapik 2017, 17.)

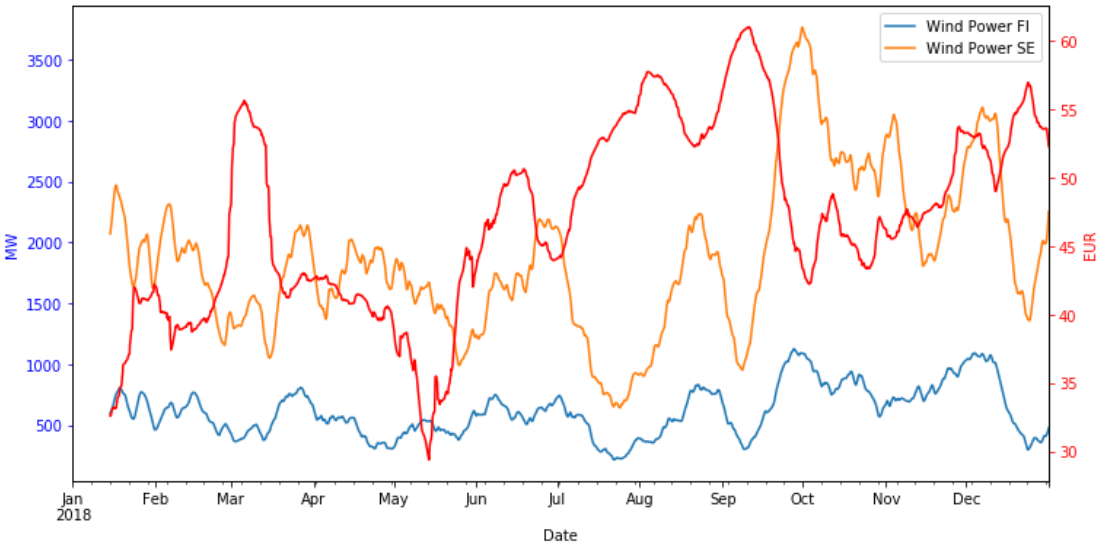

Figure 7. Wind power production in Finland (blue) and Sweden (yellow) compared to spot price (red)

The Finnish area price is sometimes more expensive than the system price set by Nord Pool, especially during the last half of 2015 as shown in Figure 8. This could be because Finland does not produce all the energy it needs and there could be congestions in the electric grid, which would raise the Finland area price compared to the system price. After 2017 the area and system prices have remained mostly the same.

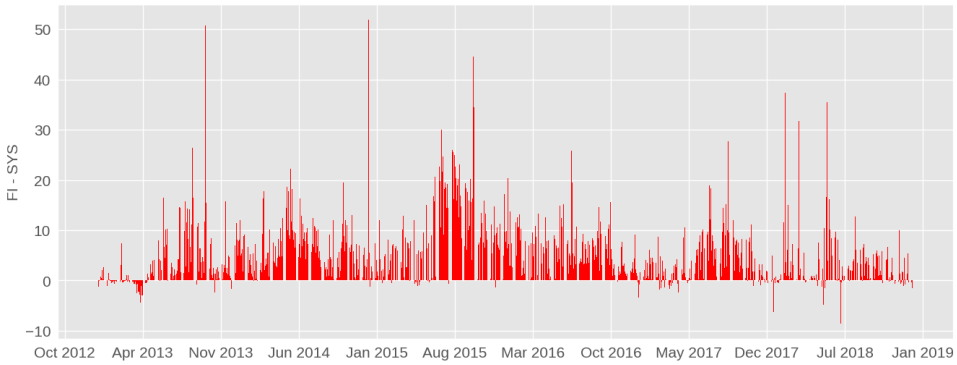

Figure 8. How Finnish area price compared to system price in 2013-2018

1.7 Previous research

Voronin researched models that could predict price spikes in Nord Pool Elspot market in his dissertation called “Price spike forecasting in a competitive day-ahead energy market”. He was able to create an algorithm that selects relevant features for spot price prediction and then build a model that forecasts demand and spot prices in the day-ahead market. (Voronin 2013, 136.)

Knapik in his research paper “Modeling and forecasting electricity price jumps in the Nord Pool power market” finds the main price drivers in the intraday market and creates three models to forecast positive and negative price spikes. (Knapik 2017, 1-2.)

Peljo’s Master’s thesis titled “Futures Pricing in the Nordic Electricity Market” studies the forecasting ability of the futures over the spot price. OSL regression analysis conducted in the thesis shows that above average stored energy in water reservoirs and hydro plants has a negative impact on spot prices and below average values have a positive impact on them. According to the thesis, rising coal prices and electricity demand also caused the spot price to rise. The futures also showed how the future spot price might behave. (Peljo 2013, 2.)

2 Data analytics

2.1 Time series

The definition of a time series is a sequence of observations taken sequentially in time. An example of this would be an hourly observation of the temperature in an airfield, which is observed through multiple years and recorded to a database. Time series can be continuous or discrete. In a discrete time series, there is a fixed interval between observations, in a continuous time series this is not the case. The time series data that will be observed in this thesis are discrete and the interval between observations will vary from three minutes to a day. (Box, Jenkins, Reinsel & Ljung 2015.)

Time series data can be divided into components. The time series has a positive or a negative trend, it can have a seasonality component that can occur daily or differentiate summer and winter seasons, there can be some cyclical movements that occur outside of seasonality and there is most likely random noise or unexpected variations. Mathematically time series can be expressed as a sum of these components. (Brownlee 2016.)

Decomposing the time series data is used to describe whether the trend in the time series is linear or exponential. This is achieved by decomposing the observed values into a trend, seasonal and residual values. If the trend grows linearly, then the time series is linear and if the trend grows exponentially, then the time series is exponential. (Hyndman & Athanasopoulos 2018.)

2.2 Forecasting

Forecasting is predicting events or values in the future and it is used in many different industries to influence decision making. Forecasts range from short-term few days forecasts to long-term forecasts that forecast multiple years. (Montgomery, Jennings & Kulahci 2015, 2.)

Forecasting techniques can be divided into quantitative and qualitative techniques. Qualitative forecasts are subjective and usually made by experts in some field, so the forecast is based on experience and knowledge about the subject. Quantitative forecasting means that historical data is utilized to create a forecasting model. (Montgomery, Jennings & Kulahci 2015, 4-5.)

There are two mathematical models that can be used to forecast future values, deterministic and stochastic models. A deterministic model requires that there are no unknown factors. For example, if one needs to calculate the acceleration of an object pushed with force F in a vacuum, it can be calculated with the formula $a = \frac{F}{m}$. If the phenomenon to be described has unknown factors in it, then the forecast is calculating a probability that the future value is between two limits, which means that the

úsed model is a probability model, also known as a stochastic model. (Box, Jenkins, Reinsel & Ljung 2015.)

The forecasts have different forms; the best estimate of a future value would be called a point estimate or a point forecast. In stochastic models, the forecasts will almost always be wrong, so it is important to calculate a forecast error for the model and provide a prediction interval, where the future values could land. The forecasts also have a horizon, which indicates how far into the future the model is forecasting values. (Montgomery, Jennings & Kulahci 2015, 5-6.)

2.3 Autoregressive model

Autoregressive models can be used to forecast future values assuming that the previous values are useful at determining future values. This is called autocorrelation, since the correlation is calculated between some variable's current values and past values. The formula to calculate a forecast is described in Formula 1:

$$x_{t+1} = b_0 + b_1 * x_{t-1} + b_2 * x_{t-2} \ldots + b_n * x_{t-n} \quad (1)$$

where t = current time

b = coefficient

The coefficients are optimized during the training phase of the model. (Brownlee 2017.)

2.4 Exponential smoothing

Exponential smoothing is a method where a time series is smoothed by decreasing the weight older values have and increasing the weight the newer the values are. This behaves similarly to a moving average, where the forecast is calculated by giving all the previous values the same weight $\frac{1}{N}$. In exponential smoothing, the weights

given to older values are determined by calculating a smoothing parameter. The different exponential smoothing methods are called single, double and triple exponential smoothing. The single exponential smoothing does not handle time series with trends well, so double exponential smoothing, also known as Holt's linear method, introduces an equation to take into account the trend component. The triple exponential smoothing, also called the Holt-Winters method introduces a third equation where a seasonality component is added to the method. (e-Handbook of Statistical Methods 2012.)

In Holt-Winters method, the forecast is calculated with an additive or multiplicative method. The additive method is used when the seasonal component of the time series stays constant, multiplicative if the component is not constant. The equation in these methods differs, in additive method in Formula 2 all the components are added together, while in the multiplicative method in Formula 3 the smoothed observation is added with the trend component and then multiplied by the seasonal component. (e-Handbook of Statistical Methods 2012.) (Hyndman & Athanasopoulos 2018.)

$$y_{t+m} = S_t + mb_t + I_{t-L+m} \quad (2)$$

$$y_{t+m} = (S_t + mb_t)I_{t-L+m} \quad (3)$$

where S_t = the smoothing component

b_t = the trend component

I_t = the seasonal component

t = current timestep

m = forecast horizon

2.5 Prophet

Prophet is an open source software published by Facebook and available as a Python package. Prophet is used to forecast from time series data, and it uses an additive model. The model is formed with components described in Formula 4. In addition to daily, weekly and yearly seasonality the model can estimate holidays effect on the time series. The seasonality component is similar to the one used in exponential smoothing. The advantage of Prophet is that good forecast results can be achieved with default parameters, and engineers with domain expertise can further tune the parameters to make the model produce better results. (Taylor & Letham 2017.)

$$y(t) = g(t) + s(t) + h(t) + \in_t \quad (4)$$

where g(t) = trend function

s(t) = seasonality

h(t) = holiday function

$\in_t$ = error term

2.6 Data manipulation

When dealing with time series, the data often needs to be manipulated in order for it to be used in forecasting or machine learning models. The different datasets may be in different frequencies, some may have observations every hour and some only weekly. Usually time series datasets are not perfect. They may have some hours or days of data completely missing or some timestamps do not have an observation recorded with them. The simplest method to deal with the missing values is to fill them in with a previous observation, which is called forward filling. Vice versa, if the missing value is filled in with the subsequent observation, this is called backfilling. (Cady 2017, 249.)

When a time series dataset is converted to a different frequency, it is called resampling. If a dataset where observations are recorded every hour is converted to a daily frequency, this is called downsampling, because the new dataset contains less information that it did previously. There are plenty of choices of what method one can use to downsample one's data, one can calculate a mean of all the observations recorded in a day and use that, or one can get the minimum or maximum value for the day and use that; it all depends on the use case. If a dataset with only daily observations is transformed into an hourly frequency, this is called upsampling. After upsampling, a method to create new values that would logically fit into the dataset is needed. The usual methods are to either simply pad the daily value into every single hour of the day or use linear interpolation to linearly change the values during the day. Padding values can be harmful when dealing with machine learning algorithms, because after the day changes, the values suddenly change to a new value: In linear interpolation, the values gently change during the day to the next day's value. (Cady 2017, 250-251.)

To visualize the differences between padding and interpolating, below in Figure 9 is a small sample size of the raw dataset with weekly values of hydro reservoirs in Finland, Norway and Sweden.

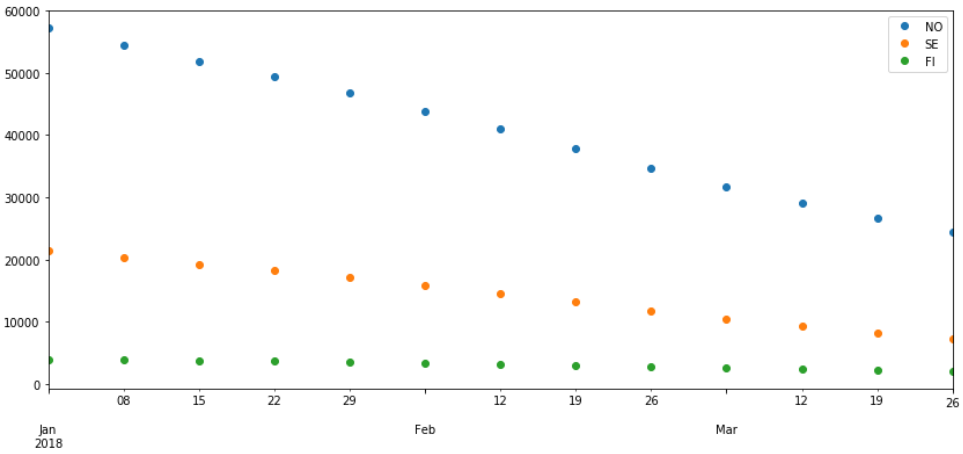

Figure 9. A subsample of the raw hydro reservoir dataset with observations in weekly frequencies

The raw data needs to be resampled to an hourly frequency. When resampling the raw data with padding, if the values are rising or declining, the resampled values will form a staircase-like line as illustrated in Figure 10.

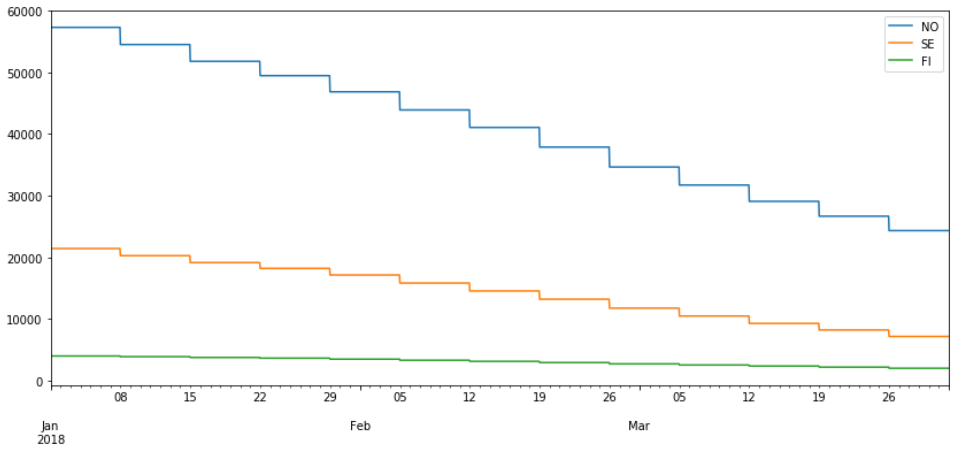

Figure 10. Weekly frequency interpolated to hourly frequency with forward filling

When raw data is resampled with linear interpolation, the individual dots are connected, forming a straight and smooth looking line as seen in Figure 11.

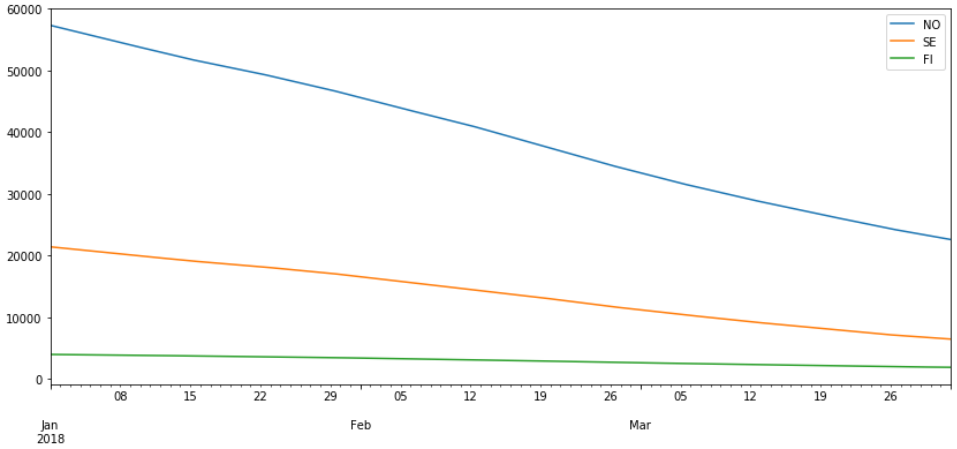

Figure 11. Weekly frequency to hourly frequency with linear interpolation

2.7 Data quality

Consumers of data expect that the data suits their needs. This can be generalized to measure data quality, where the quality is defined by how well the data serves the needs of the data's user. (Sebastian-Coleman 2013, 39-40.)

To measure the quality of the time series data that is later to be analyzed and used as training data for machine-learning algorithms, the following metrics can be used (Sebastian-Coleman 2013, 47-48.):

- Completeness: here are no null values or missing timestamps.
- Timeliness: Observations are recorded on schedule and there are no delays.
- Validity: All the recorded observations are within range of acceptable values. For example outside temperature observations would not be below -90 °C or above 57 °C, since those are the coldest and hottest temperatures ever recorded.
- Integrity: How accurately are the observations recorded and are they automatically rounded up or down?
- Consistency: How consistent would for example the temperature recordings be in 2013 compared to 2014? If there is a large difference in mean or standard deviation, something would be wrong in the dataset.

3 Machine learning

3.1 Decision trees and random forest

Random forest is a collection of decision trees. Decision trees are basically flowcharts, where every node in a tree has a question about a feature in the dataset. In case the feature is numerical, the question asks if the feature is above or below some threshold. The answer to the question then leads to a child node. (Cady 2017, 101-102)

When the decision trees are trained on a dataset, the most important variables are put onto the top of the tree and lesser variables are used as child nodes. The problem with using only decision trees is that the top-level decisions of the decision tree have a major impact on the outcome. If the decision tree is overfit during training,

then the top-level questions may be absurdly specific to the point where they cannot be generalized. This problem is mitigated with the random forest method, where the final outcome is the majority vote of the decision trees in case of classification or the mean in case of regression, which eliminates the effect of overfit trees on the outcome. This also means that less impactful features or datasets with a lot of noise do not have an impact on the random forest's results. (Brink, Richards & Fetherolf 2017, 74.)

A useful application of random forests is that it is possible to get a feature importance score after training the random forest. This feature importance score values the different features on how much they affected the prediction outcome. This can be used to determine the importance of some features, though the scores cannot be taken at face value. (Cady 2017, 103-104.)

One way to calculate feature importance is Gini importance, which is used as default in Scikit-learn's library. According to Parr, Turgutlu, Csiszar and Howard (2018) the Gini importance is biased and doesn't give an accurate presentation of the feature's importance. Gini importance in datasets where the scale of the data varies a lot may not be realiable. Parr et al. propose using permutation importance. In permutation importance, a coefficient of determination (R^2) score is calculated with a validation set. Then, new R^2 scores are calculated without one feature available and the scores are compared. The feature is removed from the test data, so the random forest model does not have to be retrained. How much the R^2 score decreases indicates how important the removed feature is. The downside of permutation importance is that it is more computationally expensive than the Gini importance. A brute-force drop-column importance would be the best way to figure out how each feature affects the outcome of the prediction. This requires training the model again after a feature is dropped, which is extremely computationally expensive and time consuming.

3.2 Artificial neural networks

Generally speaking, artificial neural networks have three layers: input, hidden and output layer. In a feed-forward network, the neurons in the input layer all have a connection to neurons in the next layer as seen in Figure 12. In this feed-forward network, the bottom layer is the input layer. From the input layer, connections are formed to neurons in the next layer called hidden layer. Each of these connections has a weight assigned to them, the neurons in the hidden layer take the connections as input and calculate an output with an activation function. Output connections of the hidden layer are connected to the output layer, where there is only one neuron and thus only one output, which could be a number for example. To simplify, the example neural networks takes two inputs, which could be for example energy consumption and outside temperature, and has a single output, which could be an estimate for the current spot price. (Buduma & Locascio 2017, 9-12.)

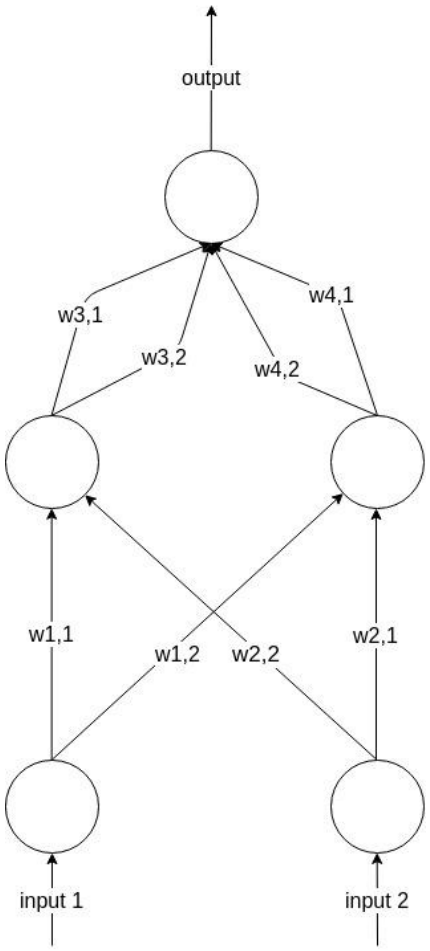

Figure 12. Example of a simple feed-forward neural network

When the neural network has been built, it will then be trained. In this example, a dataset with energy consumption, outside temperature and spot price would need to be built. The data would then be presented as a supervised learning problem, where the target variable is the spot price and energy consumption and outside temperature are features. During training, the connections weights would be updated in order to get outputs as close to the actual spot price as possible. This optimization is performed by optimizers, the most popular being stochastic gradient descent (SGD) or adaptive moment estimation (Adam). (Buduma & Locascio 2017, 17-20.)

3.2.1 Recurrent layer

A simple feed-forward neural network would not be a good choice for predicting spot prices, because the network has no memory of previous states. Recurrent neural networks implement a recurrent layer where information is shared with the neurons of the same layer. In a fully-connected recurrent layer connections are formed between all the neurons in the layer and also a connection with the neurons themselves. A fully-connected recurrent layer with n amount of neurons would then have n^2 amount of connections. (Buduma & Locascio 2017, 174.)

3.2.2 Long short-term memory

The idea behind a long short-term memory (LSTM) cell is to store information by allowing only specific gates to modify the state of the LSTM cell. An LSTM is a unit consisting of a keep gate, write gate and output gate. Keep gate determines what information is kept from the previous time step. Some information stored in the cell may be irrelevant and some not. A bit tensor that is multiplied by the previous state is computed. The result is zeros and ones; if a memory cell location holds a zero, the information is discarded and if it holds one, the information is kept in the memory. In write gate, new information is written to the LSTM unit's memory. In output gate, an interpretation of the LSTM's current state is passed on as a tensor. (Buduma & Locascio 2017, 178-183.)

3.2.3 Convolutional neural network

Convolutional neural network (CNN) is a neural network architecture, where convolutional layers are implemented. The layer consists of filters, which are applied across the input data. These filters calculate a weighted sum of the input. This is a way of choosing relevant features out of the input data. Convolutions can be performed on one-dimensional, two-dimensional and three-dimensional data. (Buduma & Locascio 2017, 95-97.)

Pooling layers are implemented after convolutional layers. Their purpose is to sharpen the features found by convolutional layers and reduces the dimensionality

of feature maps. In pooling, the input is sliced into a grid with cells and an operation is performed to each cell. Most popular pooling methods are max pooling and average pooling. In max pooling, the highest value from each cell is taken and the rest are ignored, while in average pooling the mean of all the values in the cell is calculated. After pooling, fully-connected layers are implemented to translate the features into an output. (Vasilev, Slater, Spacagna, Roelants & Zocca 2019)

3.2.4 CNN-LSTM

A neural network architecture that combines convolutional layers and LSTM cells is proposed by Donahue, Hendricks, Rohrbach, Venugopalan, Guadarrama, Saenko and Darrell (2016), where features gathered by CNN are given to LSTM units to produce a sequence with varying length. The convolutional layers can be applied to one-dimensional time series data to gather features and LSTM cells enable long-range learning from sequences. The architecture in the paper is used to generate captions for images.

3.3 Ensemble learning

When multiple neural networks are trained, each will have different weights assigned to them in the neurons and predict different values, because the training algorithm is stochastic. Predictions of multiple neural networks could be combined to create a new prediction that could have better results. This is called ensemble learning, and there are different methods on how to the predictions are combined. (Brownlee 2018b.)

Combining the predictions of multiple neural networks is called a “committee of networks”. The neural networks can be trained with different datasets, different hyperparameters etc. to create models that capture different elements of the problem that needs to be solved. A requirement for the models is to be better than baseline models, otherwise their contribution to the committee may be detrimental. There is also the question of how the predictions are combined. A simple vote or mean of the

predictions could be enough. It is also common to use another machine learning algorithm as a “metalearner”, which would assign weights to the predictions of different neural networks in order to get the best result. (Brownlee 2018b.)

3.4 Feature scaling

When working with machine learning algorithms and datasets that are recorded in different units, feature scaling is needed in order for most algorithms to perform correctly. Some algorithms for example use Euclidean distance, where the distance is calculated with Formula 5.

$$x = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \quad (5)$$

If the data was not scaled and p-values had a range of [1,10] and the q-values [10000,20000], the calculated distance would be vastly different than if the data was scaled and all values would be in the range of [0,1]. Another example is gradient descent which is used to optimize neural networks. If the data is not scaled, some weights would update faster. (Raschka 2014.)

There are two different approaches to feature scaling, z-score normalization and Min-Max scaling. In normalization, the scaled values, also known as z-scores, are calculated with Formula 6, where mean is calculated with Formula 7 and standard deviation with Formula 8. This will scale the values to a standard normal distribution, also known as Bell curve, where mean is zero and standard deviation is 1. (Raschka 2014.)

$$z = \frac{x - \mu}{\sigma} \quad (6)$$

$$\mu = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right) \quad (7)$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \quad (8)$$

The other approach is to scale all the values to a range with min-max scaling. The range is usually chosen as [-1,1] or [0,1]. This transformation can be done using Formula 9 and Formula 10 below. (Sklearn.preprocessing.MinMaxScaler n.d.)

$$x_{std} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (9)$$

$$x_{scaled} = x_{std}(r_{max} - r_{min}) + r_{min} \quad (10)$$

where r = range to where the values are scaled: $[r_{min}, r_{max}]$

The difference between the two scaling methods is visualized below. The raw four features in Figure 13 while not scaled show how spot price changes are not visible at all compared to the scale of hydro reservoir levels.

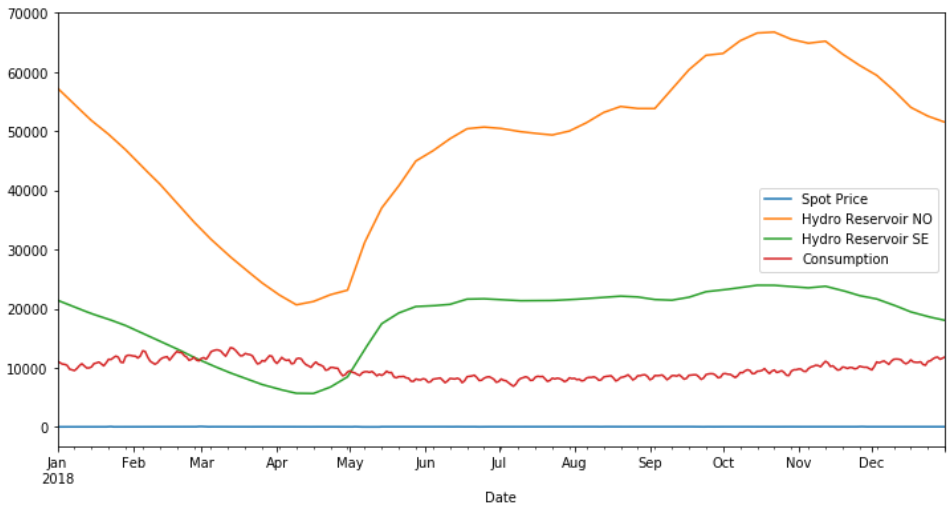

Figure 13. Raw data with no scaling

Z-score scaling is the preferred method in most machine-learning use cases, because Min-Max scaling minimizes the impact of outliers because the standard deviation of the scaled dataset will be lower than in Z-score scaling. Z-score scaling is visualized in Figure 14. (Raschka 2014.)

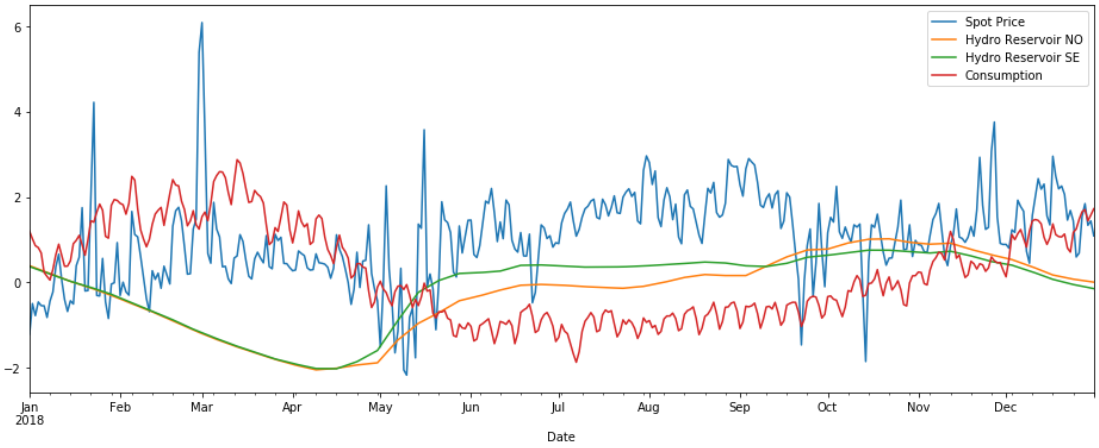

Figure 14. Four features scaled using z-score scaling

However, Min-Max scaling has its uses in scaling image data that is always in range of [0,255] and scaling data to neural networks, where the preferred range is [0,1]. The difference to z-score scaling can be seen in Figure 15. (Raschka 2014.)

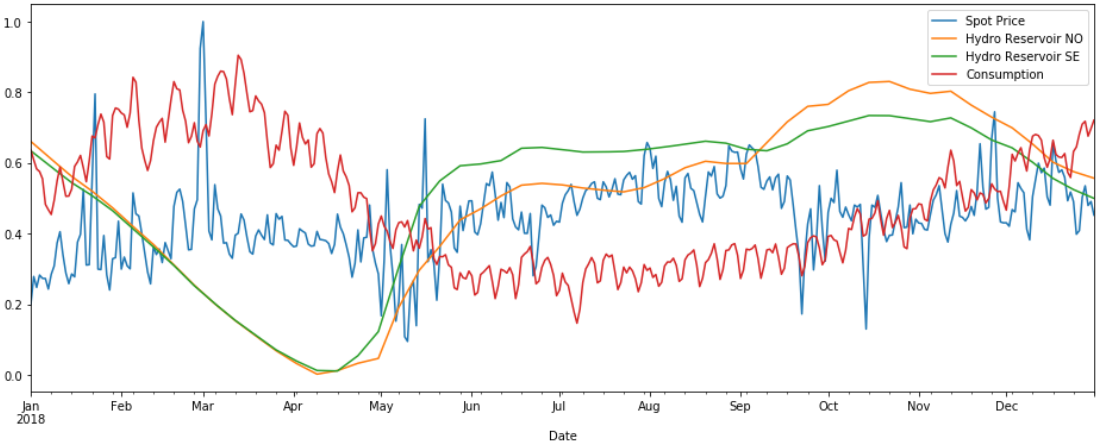

Figure 15. Four features scaled using min-max scaling between range [0,1]

3.5 Feature engineering

In feature engineering, mathematical transformations are used to create new features from raw data, that could potentially enhance the accuracy of a model. Some

useful features that could be extracted from classical time series data are mean, standard deviation, outliers and distribution. These statistics can be measured in different windows, for example the mean of a feature calculated from time t to t-24. This would be a rolling mean, which in an hourly frequency time series would tell the mean of the day before. (Brink, Richards & Fetherolf 2017, 107.)

Domain expertise is another way to feature engineer. For example, if one had to predict ice cream sales, then the outside temperature would most likely be a good feature to have. Domain expertise in this case would be compared to common sense; however, in more complicated problems experts from the problem field with domain expertise should be consulted in feature engineering. (Brink, Richards & Fetherolf 2017, 109.)

4 Forecasting the day-ahead hourly prices

4.1 Workflow

In this chapter, the historical data of spot prices is first analyzed. After the analysis, a baseline forecasting model is created. Forecasting models will first be trained on history data of spot prices alone and compared to the baseline model. Afterwards, a dataset with multiple features will be created and new models trained. In the end of the chapter, trained models will be combined into a stacking model.

4.2 Tools and operating environment

Data analysis was performed using Python and its external libraries. Python was chosen because the syntax of the language was already known. Another choice would have been R, which is an older language that is also popular among data scientists. Python is generally used in data analysis and machine learning use cases because it is an open source language, has an easy to learn syntax and the open source packages help with data manipulation, machine learning et cetera. Jupyter Notebooks were

also used to execute Python code and to visualize results. In Jupyter Notebooks, Markdown and Python code can be written and executed in cells. These cells can be executed in any order, and the variables assigned in the cells stay in memory. This helps debugging and processing new unknown data. The ability to write Markdown between cells helps code documentation. Pandas 0.24.1 version library and its data frames were used to combine data into datasets. Data visualization was performed with Matplotlib 3.0.2 and Plotly 3.6.0. Numpy 1.16.1 was used in data manipulation. Scikit-learn's machine learning library version 0.20.2 was used to train machine learning models. Neural networks were trained with Keras framework, which is built-in Google's Tensorflow 1.12.0 library. Statsmodel's library was used for time series analysis and for creating autoregression models. The operating environment was a desktop personal computer with Ubuntu 18.04 operating system installed. The required calculations to train neural networks were performed on a graphics card unit, which was Nvidia's GTX 1070. The driver version during neural network training was 415.27.

4.3 Analyzing historical market data

The market data of spot prices was analyzed by dividing the time series into three components: trend, seasonal and residual. This was achieved by Statsmodel's Python library using the "seasonal_decompose" function, which requires a time series to decompose and a frequency as parameters. The frequency is used to extract seasonality. Since the dataset is in an hourly frequency, the daily seasonality would be given by frequency value at 24. The trend, seasonal and residual components are then available as attributes. The seasonal component of the spot prices is visualized below in Figure 16.

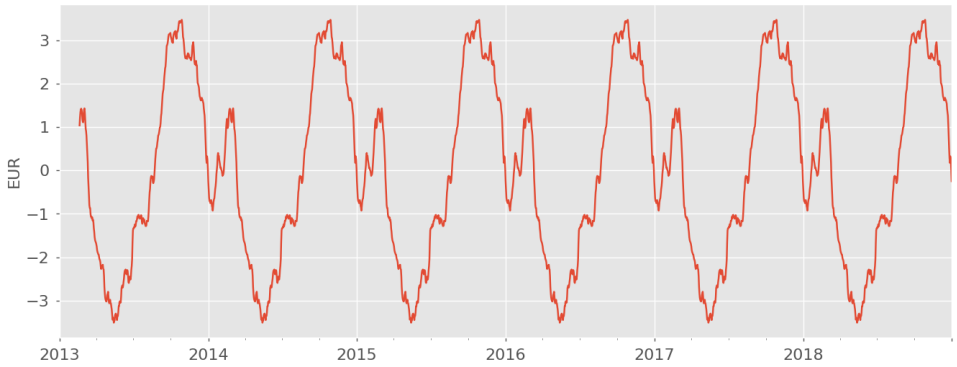

Figure 16. Spot price seasonality

The additive or multiplicative nature of the time series can be determined from the seasonal component. The component has remained constant through the years, which means that the time series is additive.

Daily seasonality

Nord Pool has split the consumption during the day into three parts. From midnight to morning 08:00 is called “Off-peak 1”. The Time period between 08:00 to 20:00 is called “Peak” and the rest of the day from 20:00 to midnight is “Off-peak 2”.

To verify Nord Pool’s definitions, the frequency in the “seasonal_decompose” function was defined as 24 for observing the daily seasonality changes. On a normal workday after five in the morning, the spot price begins to rapidly rise, until it reaches peak value at 09:00 as visualized in Figure 17. The spot prices come down until 15:00, when the spot prices reach a second peak at 19:00, which is slightly lower than the first peak.

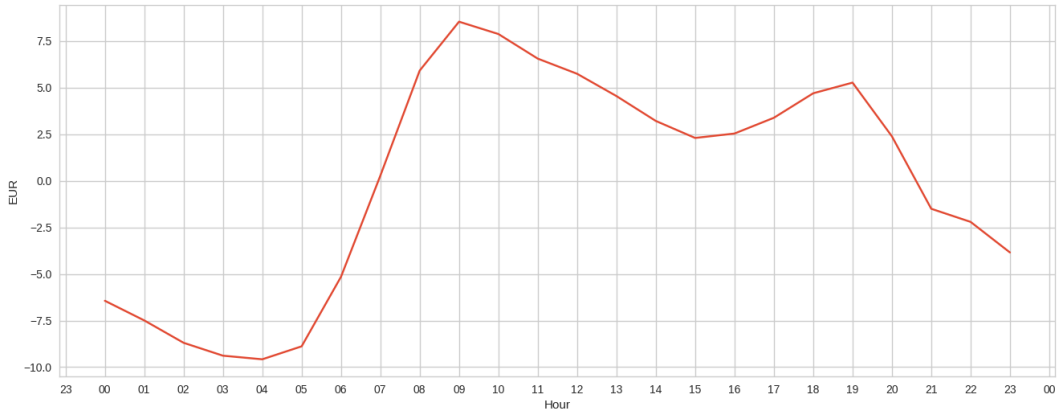

Figure 17. Daily changes in spot price.

Weekly seasonality

Seasonality value is highest at Monday morning, meaning that it is the time of the week when the spot price is usually most expensive as seen in Figure 18. Other work-days morning peaks and evening peaks stay mostly at the same level.

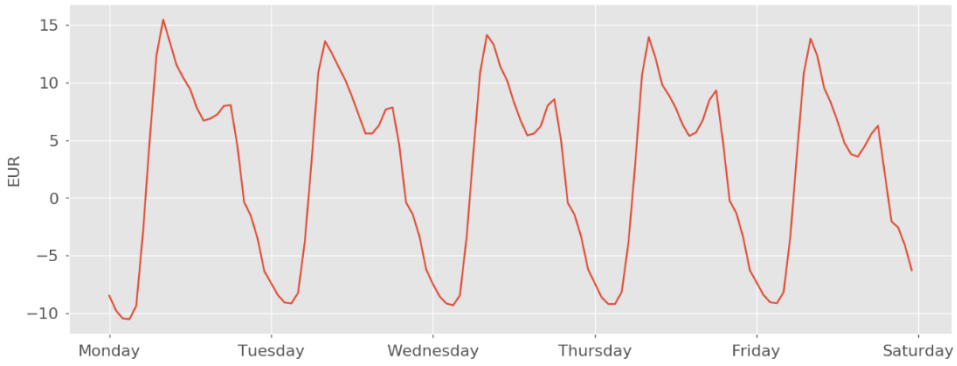

Figure 18. Monday to Friday daily seasonality, normal working days

There is a major drop in spot prices at weekends compared to workdays. The second evening peak is also higher than the morning peak as seen in Figure 19. This is most likely because a lot of stores and factories are closed at weekends and especially on Sunday.

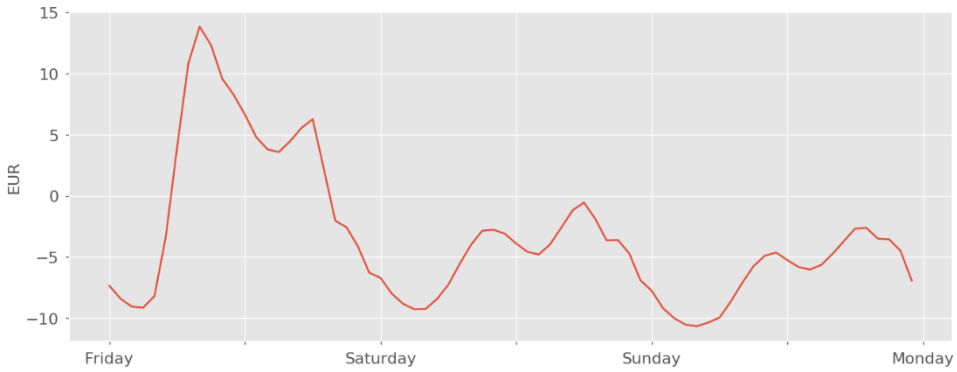

Figure 19. Friday to Sunday weekend seasonality

Yearly seasonality

The start of December and middle January are when the spot price is at its highest during the winter as illustrated in Figure 20. This is most likely because of cold temperatures. Interestingly, there is a sharp decline in spot price during the end of December, most likely because of holiday season and industry closing for a break. The spot price remains low in spring and summer, with summer's lowest prices happening at the beginning of May.

Figure 20. Yearly seasonality

Plotting the trend component visualizes well how electricity has been cheapest in 2015 and has then been increasing steadily as illustrated in Figure 21.

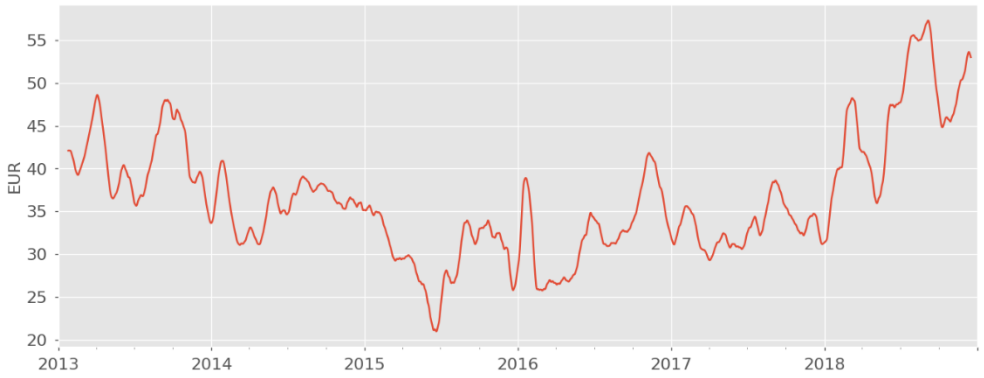

Figure 21. Spot price trend between in 2013-2018

4.4 Framing the problem as a machine learning problem

The objective of the machine-learning models is to forecast spot prices in the future. This means that the machine-learning problem is a supervised problem, because the target variable is the spot price, and the training data in this project is the history of the spot price enriched with other features. (Brink, Richards & Fetherolf 2017, 58.)

The machine-learning models will predict numerical values, which means that the problem is a regression problem and the models will be called regressors. (Brink, Richards & Fetherolf 2017, 68.)

The models can either forecast the next timestep, which in this case is the next hour, or multiple timesteps at once. Forecasting the next timestep would be singlestep forecasting and forecasting multiple timesteps would be multistep forecasting. The trained models will be mostly multistep models, since the Elspot day-ahead market for the current day is calculated from bids and offers from the previous day, the singlestep models would not be of much practical use.

If the model needs to forecast e.g. a month of spot prices and the model was trained to forecast 48-hour timesteps, a rolling forecast method will be implemented. In rolling forecasts, the example model will first calculate a 48-hour forecast from an input

sequence. The forecast values will then be appended to the input sequence, and a new 48-hour forecast will be calculated from the new input sequence. This process is repeated, until the forecasts reach the month's end.

The models are trained either with only the spot price history, which makes the model univariate or with multiple features, which makes the model multivariate. This distinction matters, because some machine-learning methods only work on univariate data and often the multivariate dataset needs to be reshaped or otherwise manipulated to work in some machine-learning methods.

How the time series data is shaped into a supervised learning problem for neural networks will be described later, when neural networks models are created for forecasting.

4.5 Forecast accuracy evaluation

The accuracy of forecasting methods and trained forecasting models will be measured on a test dataset. The test dataset will be separate from the training data and will be a continuum of the training data timewise. The test dataset will be ten weeks of data before the end of the year 2018. This means that the test data is from October 22 to December 31 as shown in Figure 22. (Hyndman & Athanasopoulos 2018.)

Figure 22. Training and test data split

The accuracy of the model can be determined on how well a model performs on the test data. The performance can be measured with error metrics. Two scale-dependent errors that were used to measure model performance were mean absolute error (MAE) and mean squared error (MSE). In mean squared error, forecast values that

are further away from real values are punished more heavily than in mean absolute error. Another error metric that is used is root mean squared error (RMSE), which is calculated by taking a square root from mean squared error's result. MAE will be calculated by Formula 10 and MSE by Formula 11. Scale-independent metrics can be used to compare datasets at that are not on a same scale or when the test datasets are not identical. Mean absolute percentage error (MAPE) is similar to MAE, except that the error is presented as a percentage. MAPE is calculated with the equation in Formula 12. (Willmott & Matsuura 2005.)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad (10)$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (11)$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| * 100\% \quad (12)$$

where y = Actual value

$\hat{y}$ = Predicted value

Coefficient of determination (R^2) is also a good indicator of a model's performance. An R^2 score is calculated with Formula 13. Total variation (TV) is calculated in Formula 14, and residual variation (RV) is calculated in Formula 15. If the R^2 score is close to zero, it is the same as the function returning the mean of the dataset. This gives a good indicator of the model's performance, since if it is close to zero then the model does not perform better than simply predicting the next expected value. Since the total and residual variation is subtracted from one, the skill of the model increases as it nears the number one. (Cady 2017, 159.)

$$R^2 = 1 - \frac{RV}{TV} \quad (13)$$

$$RV = \sum_i (y_i - \hat{y}_i)^2 \quad (14)$$

$$TV = \sum_i (y_i - \bar{y})^2 \quad (15)$$

where $\bar{y}$ = mean of real values

$\hat{y}$ = forecast values

When the test dataset is longer than the amount of timesteps that the models forecast, a walk forward validation method will be implemented. Like the rolling forecast method, the model will go through the test dataset in timesteps, however after every timestep, the model will forecast with real values from the test dataset, rather than from its own forecasts. This method simulates real-world use and allows for the model's accuracy to be evaluated on the whole test dataset. Below in Figure 23 is a Python implementation for walk-forward validation to an autoregression model, where history is the training data and the test data that the model has already forecast on.

```
import numpy as np
def ar_forecast(history,forecast_horizon):
    history = np.array(history)
    model = AR(history).fit()
    yhat = model.predict(len(history),len(history)+(forecast_horizon-1))
    return list(yhat)
```

Figure 23. Forecasting the length of forecast horizon

The function above is called during the model's training. The forecasts that the model makes from the history data is saved to a list, and then the real values from the test dataset to history data. The amount of values to be added is determined by "forecast_horizon", where if the model forecasts the next 48 hours, "forecast_horizon" would be equal to 48. A full implementation of the walk-forward validation method for the autoregressive model is visualized in Figure 24.

```
predictions_ar = []
history_ar = [spot_price for spot_price in training_data]
hour_counter = forecast_horizon
for i in range(int(len(test)/forecast_horizon)):
    yhat_list = ar_forecast(history_ar,forecast_horizon)
    for yhat in yhat_list:
        predictions_ar.append(yhat)
    for x in range(hour_counter-forecast_horizon,hour_counter):
        history_ar.append(test.iloc[x])
    hour_counter += forecast_horizon
return np.array(predictions_ar)
```

Figure 24. Walk-forward validation for autoregressive model

4.6 Forecasting from Elspot price history

A simple baseline model is created. The model forecasts future values by simply repeating the previous X values, where X is the forecast horizon. The baseline model is created for 24-,48- and 168-hour forecast horizons, which means that the model's forecast will be the same as spot price was 24, 48 and 168 hour ago. The forecast errors of the baseline model are listed below in Table 3.

Table 3. Baseline model forecast errors

Forecast Horizon	**MAE**	**MAPE**	**RMSE**
24	5.34	9.06	10.28
48	7.47	11.74	14.59
168	7.02	11.24	13.83

Already from the baseline model forecast results it can be seen that spot price has a strong weekly seasonality, since the errors in forecast horizon 168 are smaller than the errors in forecast horizon 48 model as seen in Figure 25.

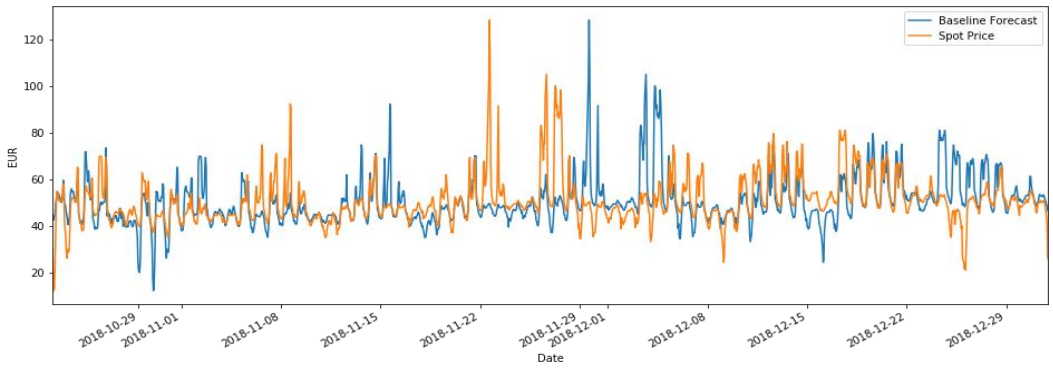

Figure 25. Baseline model forecasts

The forecast errors of new models will be compared to the forecast errors of the baseline model in order to give a good idea of how skillful the newly created model is.

4.6.1 Autoregression

An autoregression model was trained that forecasts the next 24 hours using the “AR” class in Statsmodel’s library. The model is evaluated on the whole test dataset using the walk-forward validation method. As would be expected, the model reacts a day late to the sudden upward or downward spikes in spot prices as seen in Figure 26. However, if there are no sudden spikes, the model is able to produce an acceptable forecast on normal workdays. For a simple model, it achieves quite low MAE and RMSE errors.

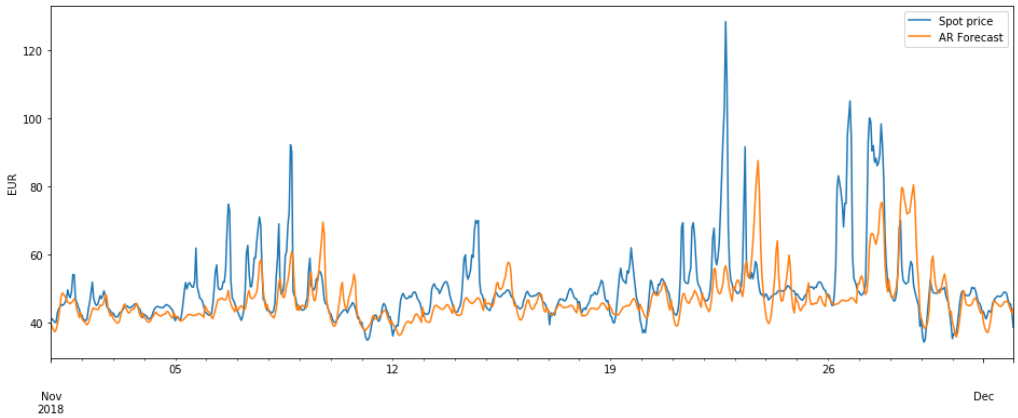

Figure 26. Autoregression model predictions compared to real spot prices

4.6.2 Holt-Winters exponential smoothing

A Holt-Winters model requires seasonal period length and the nature of the seasonality as parameters. The nature of the seasonality can be additive or multiplicative, which in this case was earlier determined to be additive. The seasonal period was set as 24, 36 and 168. Because exponential smoothing method averages the weights of past observations, the forecast's daily seasonality is affected if there were large spikes in the spot price as seen in Figure 27.

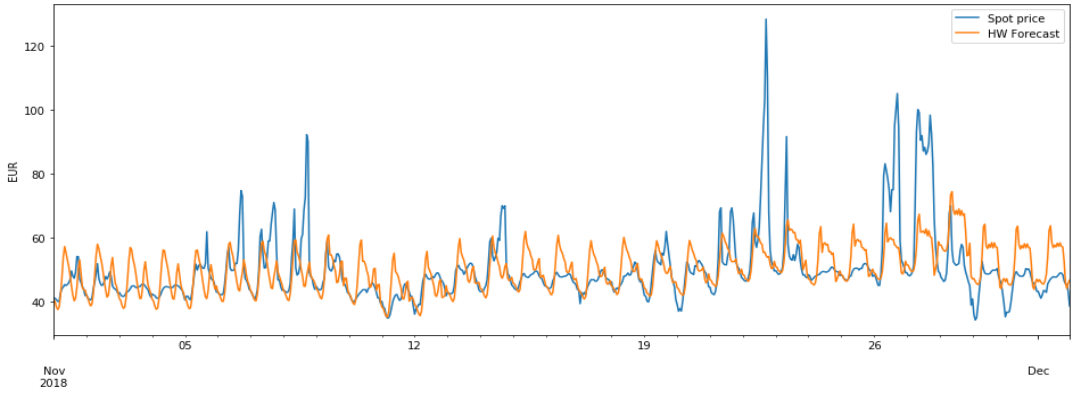

Figure 27. Holt-Winters forecasts compared to real spot prices

4.6.3 Prophet

The prophet model can be tuned by changing the following hyperparameters: changepoints, n_changepoints, changepoint_range and changepoint_prior_scale. Changepoint is when the time series trend changes suddenly. The changepoint prior scale determines how much the changepoints fit into the data. In case of the spot price, the spot price weekly trend lowers during the weekends and yearly trend lowers during the summers, at least usually. The effects of holidays can also be changed. It was seen earlier that the spot price trend lowers during winter holidays. (Van der Merwe 2018)

Prophet model forecasts from spot price history data alone, since the use of exogenous variables is not supported. The prophet model is fast to train; therefore, a grid search was implemented to find the best hyperparameters. According to Taylor and Letham (2017, 21) the model can perform worse with more data, since a longer history can mean that the model is overfit to past data that is not maybe as relevant in the future. Thus, the length of the training data was also included in the grid search.

After grid search, the best parameters for the model were found. The changepoint prior scale was at 0.05, n_changepoints at 48, holidays_prior_scale at 10, seasonality_prior_scale at 10, seasonality mode in additive and the model was only given data from 2018.

The model was made to predict on the test dataset and the forecasts were compared to actual spot prices. The model predicts spot prices on average to be 7.2 euros lower or higher than the real value. When plotting the forecast values and actual values in a line graph in Figure 28, it can be seen that the model has learned the daily and weekly seasonality well. The weakness of this model is that it is unable to predict sudden spikes in the spot price.

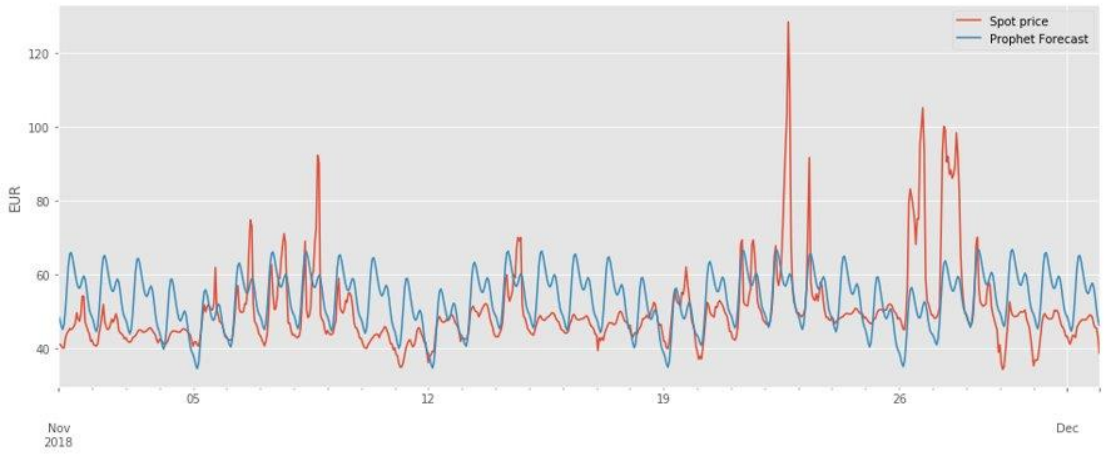

Figure 28. Prophet's forecasts and spot prices

The Prophet model provides a data frame of yearly, monthly, weekly and daily seasonality along with additive terms. These were later used as features in the new dataset.

4.7 Creating a data frame for the datasets

Data frames from Python's Pandas library was chosen as the method to combine the datasets into one table-like format. Pandas was chosen because the knowledge required to use the tool was already gathered and its premade functions make data storage and visualization easy.

Since the spot price needs to be predicted hourly, the data frame index is required to be in an hourly frequency. Pandas allows datetimes to be used as indexes. This allows easier windowing of the data, since calendar dates can be used instead of numerical values.

When datetimes are set as index, time zones and daylight savings need to be considered when combining different datasets together. Especially daylight savings presents a problem, since some datasets skip an hour when changing to summer time and add a same timestamp when changing to winter time, and other datasets do not

acknowledge the transition at all. It was deemed necessary to make all the datasets time zone-naive and convert them to Universal Time Coordinated (UTC). The summer and winter time transitions were also ignored in the time zone-naive index. The alternative approach of keeping all indexes time zone-aware was tested, but multiple problems arose when time zone-naive datasets needed to be added. Thus, the time zone-aware index was discarded.

The first variable inserted to this data frame will be the spot price history. New variables will be added to this data frame, and it is referred to data group data frame from now on in the documentation.

4.8 Gathering the variables that affect the day-ahead price

Since the spot price is calculated for every hour of the day, and the price data is available in an hourly frequency, it is necessary for all the new variables also to be in an hourly frequency. This meant that the datasets would be downloaded as hourly versions if available. If not, the datasets would be downsampled if the observations were recorded more frequently than hourly or upsampled if less frequently.

Pandas has in-built file parsers for CSV, Excel, HTML and JSON files. When a new dataset was downloaded, the values were read into a temporary Pandas data frame using the in-built parsers. The “Date” columns in the new dataset is set as index, like in the data group data frame. After the new index is set, the new dataset can easily be inserted into the data group data frame as a new column, since Pandas compares the indexes between the data frames and inserts values accordingly, as long as the indexes are in the exact same format.

An example of how the price of emission allowances was loaded into a data frame is visualized in Figure 29. In the example, a data frame is created, where the index is a datetime index with an hourly frequency. The emission allowances are read into a separate data frame, using Pandas’ “read_csv” function. The date column of the data frame is converted to datetime format and set as the index. The numerical values

were all also converted to “float64”. The values are then transferred to the column that was created first. Since the emission allowances are stocks, the dataset only has values recorded from business days and only one value per day, which is the stock closing price. This means that the data frame will have multiple null values, which need to be filled using linear interpolation.

```
def load_carbon_emission_permit_data():
    dataframe = pd.DataFrame(index=pd.date_range(start='01/01/2013', end='01/01/2019',freq='H'))
    values_df = pd.read_csv('data/eex_emissions_market/eua-price.csv')
    values_df['Date'] = pd.to_datetime(values_df['Date'])
    values_df = values_df.sort_values(by='Date')
    values_df.set_index('Date',inplace=True)
    values_df = values_df.astype('float64')
    dataframe = values_df
    dataframe = template_df.resample('H').interpolate(method='time').bfill()
    return dataframe
```

Figure 29. Function for reading emission allowance data

Many of the datasets had missing timestamps or missing values, which needed to be cleaned up. The problems found in the dataset and data cleaning operations were documented below.

4.8.1 Weather data

Finnish Meteorological Institute (FMI) has established an open data portal, where everyone can download weather related data. It is possible to choose a time period, observations and the observation station. The link for data download is at https://en.ilmatieteenlaitos.fi/download-observations#!/. Due to limitations in the service, weather data for the years between 2013 and 2018 had to be downloaded multiple times, since only 18 months amount of observations could be downloaded at once.

The downloaded weather data was downloaded from Helsinki-Vantaa airport observation station, where it observed:

- Amount of clouds
- Air pressure
- Relative humidity

- Precipitation intensity
- Snow depth
- Air temperature
- Dew-point temperature
- Horizontal visibility
- Wind direction
- Gust speed
- Wind speed

Swedish meteorological data was downloaded from Swedish Metereological And Hydrological Institute, which provides open data from a number of observation stations across Sweden. The link for download is at https://www.smhi.se/klimatdata/. Precipitation data was chosen from Ritsem A station, which is near the largest hydro power plant in Sweden, Harsprånget. The temperature data was downloaded from Stockholm-Bromma airport weather station.

The data for the years 2013 to 2018 had to be downloaded in two separate parts, because the first dataset did not include the latest three months. The missing data from the first dataset was filled by hand.

In FMI's data for some reason a week of data was missing from 2018 between the dates: July 2 2018 and July 9 2018. The missing time range was discovered, and values were downloaded again and added to the dataset by hand.

4.8.2 Svenska Kraftnät open data

Svenska Kraftnät (SVK) is the state-owned electricity transmission grid operator in Sweden. SVK has made production and consumption datasets public and open for download at https://mimer.svk.se. The observations are recorded in hourly frequencies, and most of the records start in 2015, except for consumption profile, where records start at 2013. There is no option to select a time zone for the dataset, nor is the time zone listed anywhere on the website, so it is assumed that the observations are in Sweden's time zone, which is UTC+1.

Svenska Kraftnät datasets had missing values for the whole month of August in 2015. To fill the missing values, a mean was calculated from the previous and next month for each hour in September. There were also huge outliers in the data, which lasted for an hour or two. The outliers were replaced with the median value of the whole dataset, because machine-learning models will most likely not learn anything useful from a couple hours of huge outliers in a dataset with tens of thousands of hours of observations.

4.8.3 Fingrid open data

Fingrid, like SVK, is the transmission grid operator in Finland. Fingrid has made its data open for public use, and the data can be downloaded via a visual interface or an API. The datasets had values recorded in hourly frequencies. The time zone was changed to UTC+0, which caused the timestamps in the CSV file to be two hours earlier than the time range specified in the visual interface. The downloaded datasets included the transmission between Finland's neighbors, bottleneck income, electricity production et cetera.

Nuclear power generation data was in five-minute frequencies and the API had a limit of how many rows can be requested at once meant that downloading six years of data all at once was not possible. A function was programmed in a Jupyter Notebook where the data is requested in smaller chunks and then combined into a large JSON file. Electricity generated by wind power was also recorded with the same frequency as nuclear power production, which meant that the same function was used to combine wind power production data to one file.

Nuclear power production and wind power production had empty rows in the dataset. In order to downsample them into an hourly frequency without null values, the values were added with a forward-fill method, meaning that the last known value is used to fill the missing values. Using this method, 102 missing values are filled into the nuclear power production column and 2147 missing values into the wind power production column.

4.8.4 Hydro reservoir data

Hydro reservoir stored energy values were available for all European countries from https://transparency.entsoe.eu/. The hydro reservoir data from Norway, Sweden and Finland were downloaded from the years 2015-2018 The graph below in Figure 30 shows how in Norway and Sweden, the stored energy value growth in 2018 is cut short compared to earlier years and decreases for a short period of time.

Figure 30. Water reservoir stored energy values in Norway, Sweden and Finland

The water reservoir data is available in weekly frequencies and was missing values. The missing values were inserted manually to the table using transmission system operator's data. In order to combine the dataset with other's that are in an hourly frequency, the water reservoir data needed to be upsampled. The missing values caused by upsampling the data were replaced using linear interpolation as shown earlier in Chapter 2.6. There were also some missing values in the beginning of the time series, since the start of the week does not line up with the start of the year 2013. The values from 1 January 2013 to 1 June 2013 were backfilled from the first available weekly value at 1 July 2013.

4.8.5 Nordic power futures

Nord Pool's financial market is handled by Nasdaq OMX Commodities, where derivatives and futures for spot prices can be traded. Since the futures are agreements to buy or sell electricity in the future, data about futures could indicate increase or decrease in spot price. Unfortunately, futures data was not available for public use, thus it could not be used as a feature in the dataset.

4.8.6 Coal and carbon emission permits

The EU has implemented an Emission Trading System designed to reduce carbon emissions. The idea is that companies causing emissions must buy emission allowances and use them to cover their emission amount. Leftover emission allowances can be saved for the future or sold to other companies. The total amount of allowances is limited by a cap that is lowered over time in order to also lower the amount of emissions. (EU ETS Handbook 2015, 4-5.)

The cost of these emission allowances has been steadily rising since 2017 along with the price of coal as seen in Figure 31, which has caused electricity prices to rise in Britain, France and Germany. Due to Germany still using coal plants to produce electricity, the price of the allowances may increase even further. (Morison & Hodges 2018.)

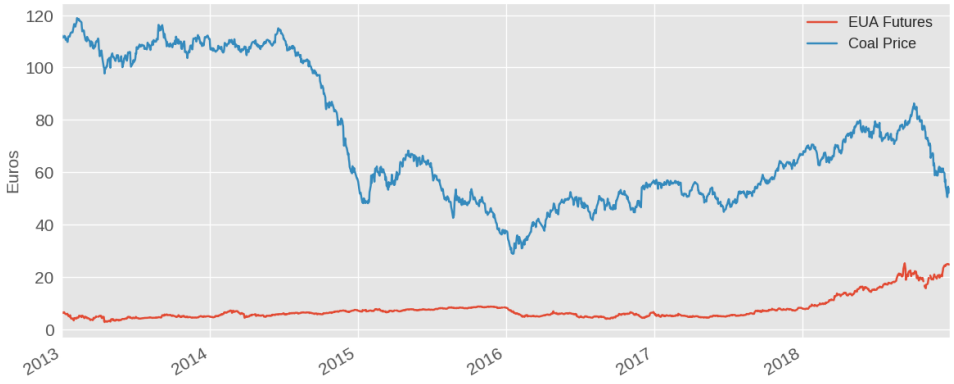

Figure 31. Coal and emission allowance futures prices since 2013.

Coal is used to generate electricity in the Nordic countries, when electricity consumption exceeds hydro and nuclear power production. This usually happens in cold winter days, when the coal plants are started up to fill electricity supply gaps. Additionally, in an event where electricity needs to be imported to the Nordic region, it is mostly imported from Germany and the electricity is likely generated by coal. (Knapik 2017, 7-8.)

4.8.7 Time

The fluctuating spot price is clearly cyclical. There is a clear 24-hour cycle of how the price usually behaves, a weekly cycle with the price being lower on the weekends and a yearly cycle with price being higher in winter times.

Simply using a number to describe the time does not convey its cyclical nature to a machine learning model. For example, the hours 0.00 and 23.00 are closer than hours 0.00 and 12.00, yet a machine learning model would think they are not, because the distance between zero and 23 is larger and the distance between zero and 12. Features where the cyclical nature of the time is conveyed properly can be created with sine equation in Formula 15 and cosine equation in Formula 16: (London 2016)

$$t_{sin} = \sin(\frac{2\pi x}{y}) \quad (15)$$

$$t_{cos} = \sin(\frac{2\pi x}{y}) \quad (16)$$

where x = time

y = length of time period

For example, if a day needs to be converted to a cyclical time, x would be the hour and y would be 24, since there are 24 hours in a day. An example of sine values for each hour are visualized in Figure 32.

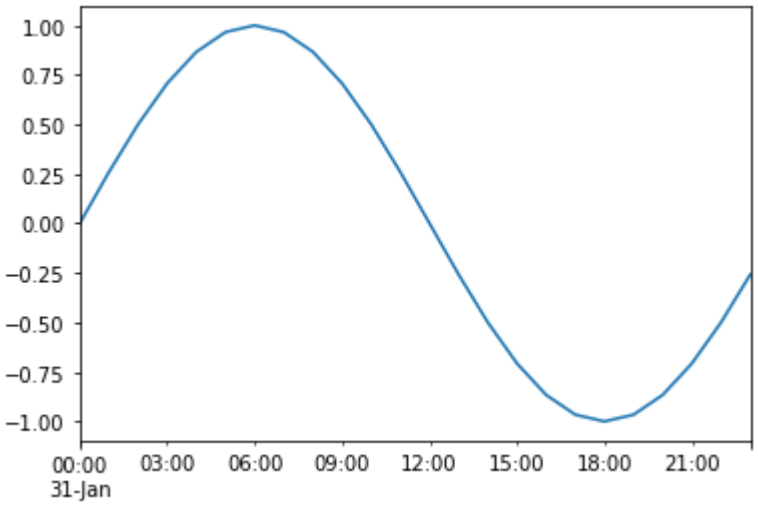

Figure 32. Hours transformed into a sine wave.

Respectively the cosine values for each hour are visualized in Figure 33.

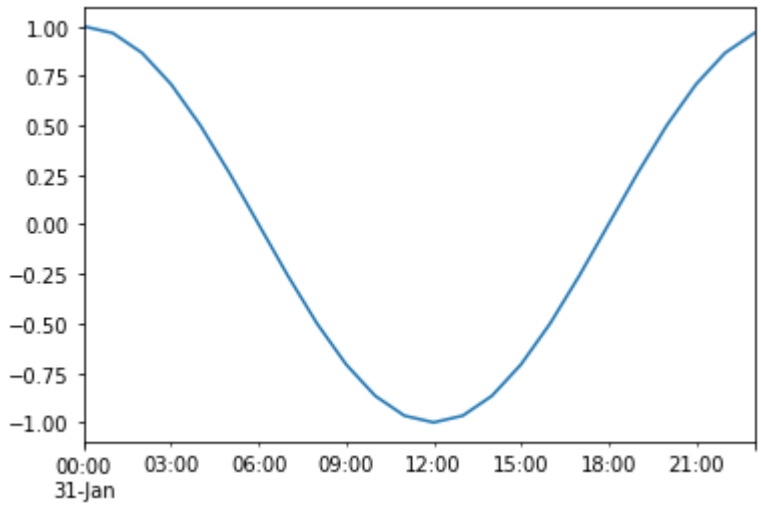

Figure 33. Hours transformed to a cosine function

When the cosine values are plotted on the y-axis and sine values on the x-axis, the result is a clock-like circle, where it can be seen that for example the dots closest to 0.00 are 23.00 and 01.00 as illustrated in Figure 34.

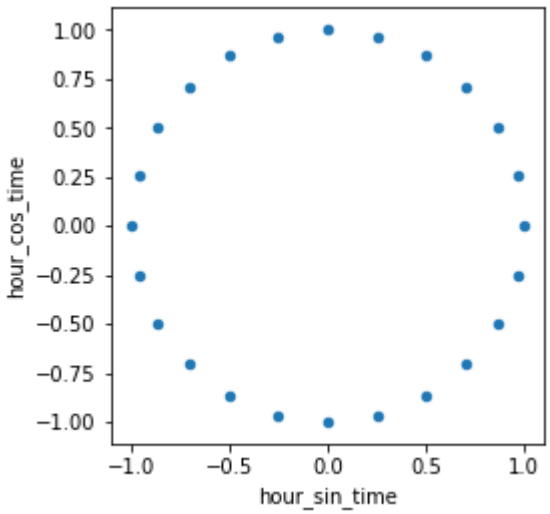

Figure 34. Hours converted to a cyclical time

The same function is used to create cyclical times for months, weeks and years. This means that now Monday is close to Sunday, January is close to December and the first day of January is close to the last day of December.

4.9 Feature engineering

Feature engineering principles are used to create new features. With time series data, rolling mean and standard deviation with varying window sizes were used as features. A rolling mean with a window size of 36 for example would calculate the mean of the latest 36 spot prices. A lagged version of spot price can be used as a feature also. spot prices lagged by one day and one week were added as features, because the baseline model already showed, how adequate forecast results can be achieved by just using the prices from day before or week before.

Having too many features can negatively affect the accuracy of the models. A list of features to be deleted was created from the result of random forest algorithm's permutative importances function. Another dataset was created from Scikit-learn's own feature importances function, in order to compare which function produced a better dataset.

4.10 Forecasting with the data group

When the data frame with multiple features was created, new machine learning models were trained, where the training data now had multiple features.

4.10.1 Random forest

A random forest was created using Scikit-learn's "RandomForestRegressor" class. While random forest is not usually used for time series forecasting, its feature importance function can be used to determine what features correlate with spot price. In order to simulate a forecasting situation, all features except the spot price needed a new lagged version of the feature, where the lag is the forecast horizon.

Without hyperparameter optimization or knowledge of spot price at previous timesteps, random forest has error values MAE at 7.71, MAPE at 18.16 and RMSE at 10.81, which means that the performance is below the performance of the baseline model. However, when a time lagged column of spot price values is created, the results improve.

To optimize the training parameters of the random forest, a random search was implemented. This meant that a wide range of possible training parameters was defined, and among 2880 possible combinations, 50 combinations were randomly selected and tested for how accurate they were at forecasting the spot price. Cross-validation method was also implemented, where a different portion of the dataset was used for evaluation. The amount of combinations to be tested and number of cross-validations were set low, because some random forests took a minute or longer to train and test. The best training parameters found by random search are listed in Table 4.

Table 4. Random forest best parameters found by random search

Parameter	Value
Estimators	280
Minimum Samples Split	10
Minimum Samples Leaf	2
Maximum features	Square root
Maximum depth	10
Bootstrap	False

These training parameters achieved a MAE of 6.63, RMSE of 10.14 and MAPE of 14.50. To further reduce loss, grid search was implemented. In grid search, all the given parameter combinations are tested. The tested parameters were chosen as values close to the best result in random search. The tested parameters are listed in Table 5.

Table 5. Random forest grid search parameters

Parameter	Value
Estimators	100, 200, 300
Minimum Samples Split	8, 10, 12

Minimum Samples Leaf	2, 4, 6
Maximum features	Square root, No maximum
Maximum depth	7, 9, 12
Bootstrap	False

The amount of different random forests that were created in this grid search was 3*3*3*2*3*1 = 162. To test all the possible combinations, the grid search took almost an hour to finish. The best parameter combination is listed in Table 6.

Table 6. Random forest best parameters found by grid search

Parameter	**Value**
Estimators	200
Minimum Samples Split	12
Minimum Samples Leaf	4
Maximum features	Square root
Maximum depth	12
Bootstrap	False

The losses with these training parameters were: MAE 6.62, RMSE 10.07 and MAPE 14.50. Considering that the loss reduction is quite minimal from random search's best parameters and the fact that grid search took almost two hours to complete, searching for the best parameters would not be worth it in production, where time and computing power availability can be a bigger factor.

4.10.2 Feature importance

Feature importances were listed from the trained random forest model. This provides a percentage for each feature meaning how much the feature affects the final prediction. The feature importances were plotted into a horizontal bar graph. To compare the random forest's Gini importances and permutative importances, both were retrieved from the same random forest model. According to Gini importance as visualized in Figure 35, the precipitation and cyclical month features had zero impact on the final prediction. The most significant features were the spot price one week ago, spot price 24 hours ago Prophet's daily seasonality and Prophet's additive terms. All these features had an importance score of 0.05 or higher.

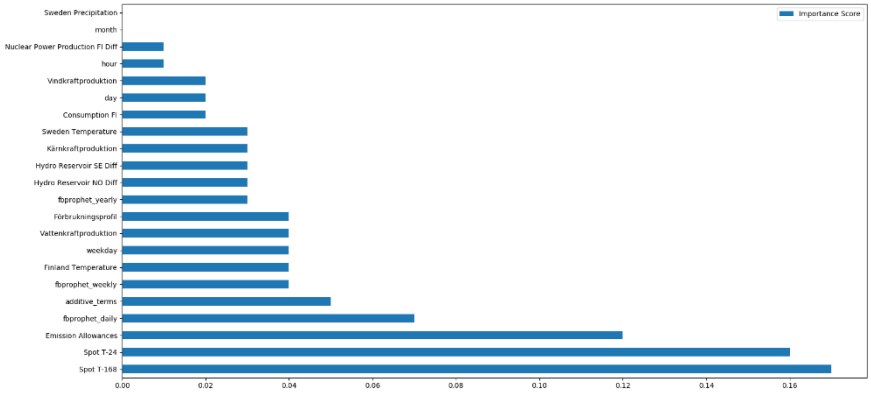

Figure 35. Random forest Gini importance scores

Permutation importance, visualized in Figure 36, calculated Prophet's daily seasonality to be the most important feature with a 0.15 importance score. Second-best feature was Prophet's additive terms, all other features were below 0.07 score. Interestingly, energy consumption in Sweden, temperature in Sweden and Finland and yearly seasonality of Prophet gained a negative importance score.

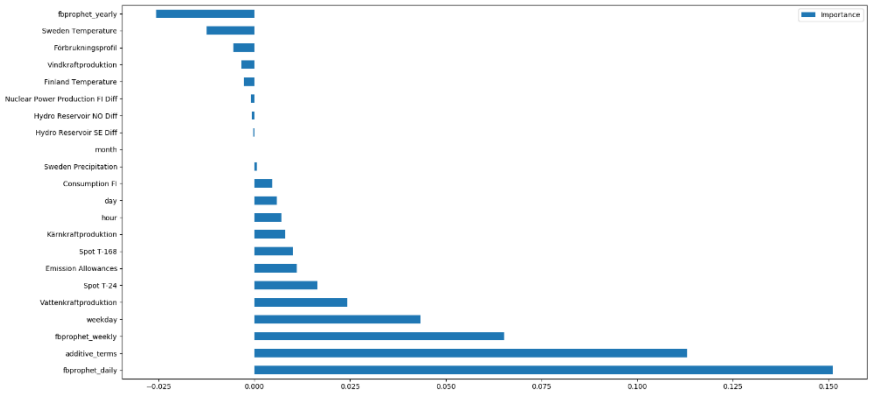

Figure 36. Random forest permutation importance scores

The results of Gini and permutation importances are vastly different. Both agree that precipitation data at its current form is not useful. However, no further assumptions regarding what features to cut can be made. On some tests, where one-hot-encoded data was included, the importance scores claimed that the one-hot encoded features had no importance on the prediction. However, when the random forest was trained without one-hot encoded features, the losses were higher. This most likely means that the feature importance score is not compatible with one-hot encoded features and may not always provide informative results.

4.10.3 Data preparation for neural networks

The sequence length in the networks will be 24, 48 and 168 which are one-day, two-day and one-week forecast horizons. To visualize the sequences and how training

data sequences will look like, below in Figure 37 is an example of sequence length 3.

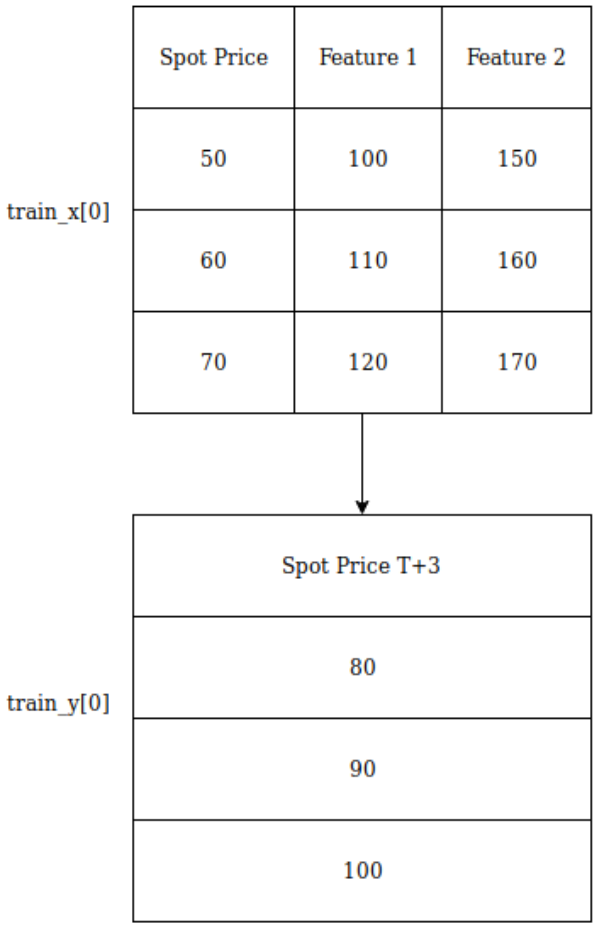

Figure 37. Training data sequence at index 0

Using this supervised learning technique, neural networks are made to learn that the spot price sequence of “50, 60, 70” and other features leads to spot price sequence “80, 90, 100”. This is repeated for every hour in the dataset, so the next element of the matrix “train_x” and array “train_y” is visualized below in Figure 38.

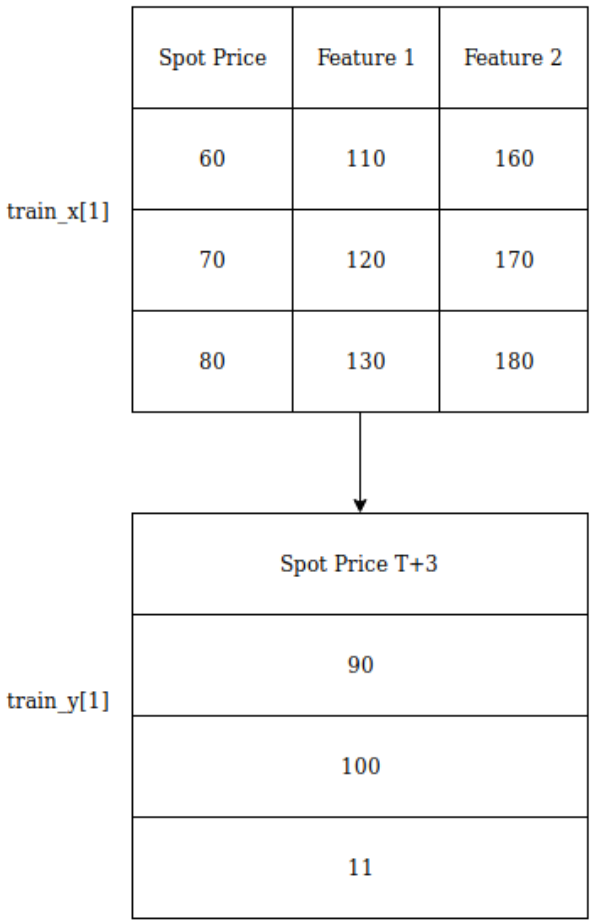

Figure 38. Training data sequence at index 1

All the variables had to be scaled in order for the networks to correctly learn the weights between the neurons. A min-max scaler from the Scikit-learn's library was used, which scaled all the values between 0 and 1. When the model predicts future values, only the future spot prices are forecast. If a min-max scaler was trained on 17 columns, the scaler requires that the input shape (1,17), which is a matrix with 17 columns and one row. However, the scaler applies scaling to each row separately, which means that NumPy library can be used to create an array of zeros, that is appended to the spot price. This allows the spot prices to be inversely transformed without requiring other values. Later during testing, changing the min-max scaling to standardization helped achieve better results, even though it is recommended for neural networks to receive data in the range of 0 to 1.

4.10.4 Training and testing framework

The networks were trained with varying hyperparameters in search for the best test results. The trained neural network models were saved and named after network type, sequence length, training dataset and a timestamp value at the end to prevent duplicate names. A checkpoint and early stopping system were implemented, where if the model's error did not improve on the validation set for 10 epochs, the training would be stopped. If the error did improve, then a checkpoint model would be saved and the previous save overwritten if the error continues to improve.

After the model's training completed, the model's performance was tested on a test dataset, which was defined in chapter 5.4. MAE, RMSE and MAPE error values were calculated by comparing the test dataset to the predictions. Error values, model's training hyperparameters and model's filename were saved to a CSV file.

The training is started by running a Python script with command line parameters. An example shell command to run the script would be:

"python LSTM_Encoder_Decoder.py --dataset data/dataset.pickle --hparams hyperparameters.txt --cnn"

This would use the "dataset.pickle" file as training and test data, and the "--cnn" argument would create a CNN-LSTM network. The training hyperparameters would be loaded from the "hyperparameters.txt" file, where the hyperparameters would be in a dictionary format. The hyperparameters can contain multiple hyperparameters if multiple models need to be trained in succession.

4.10.5 Onestep LSTM

A multivariate LSTM network was created, which means that the network uses multiple features to predict the spot price. In total there are 26 features in the training dataset. The created network uses the spot price and other features at timestep t-1 to predict the spot price at timestep t. The network is a simple network with a single hidden layer, which is a LSTM layer that is fed the input directly and that connects to

a fully-connected layer. The fully-connected layer then gives the spot price value as output. The model uses the last 48 timesteps, which means that the input shape will be 48 rows long with 26 columns, as seen in Figure 39.

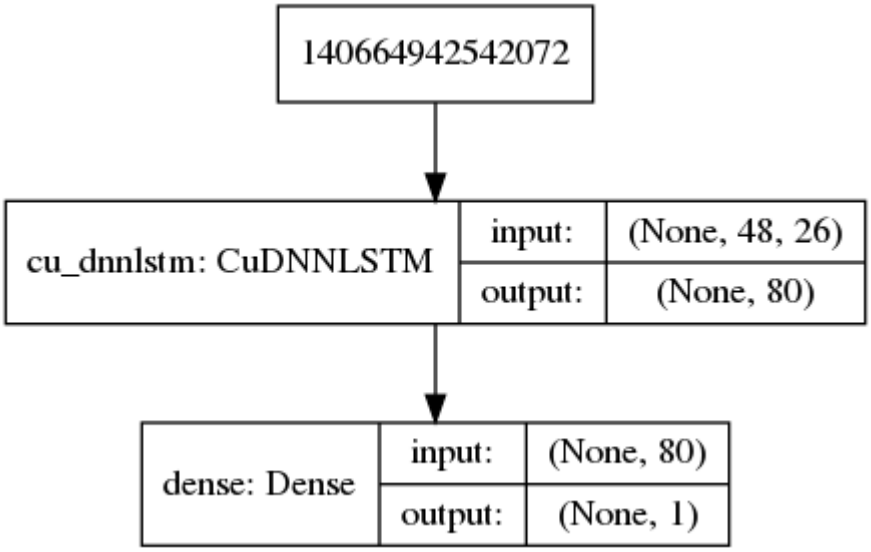

Figure 39. Simple LSTM network architecture

While the model provides good results, as can be seen in Figure 40, it is not suited well for spot price forecasting, since the buy and sell orders are sent the day before, which means that predicting one hour ahead has no real-world application. This model should be modified to make predictions more than t+1 timesteps ahead.

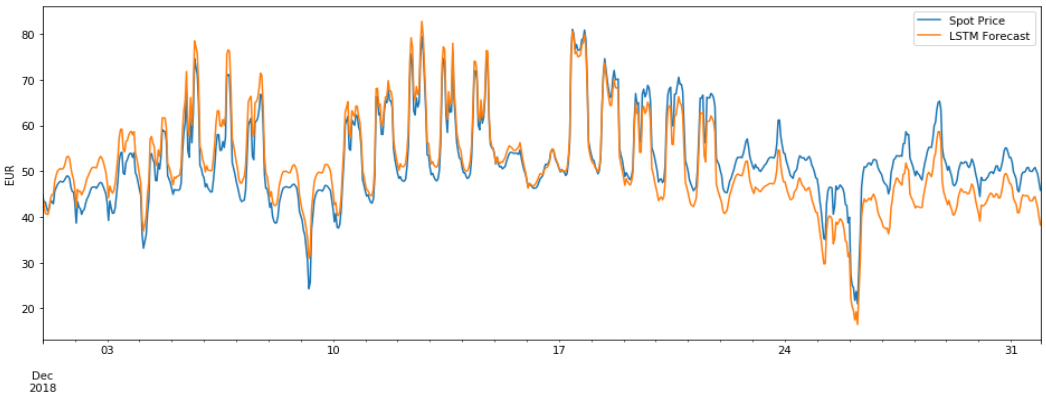

Figure 40. Onestep LSTM forecasts (yellow) and spot prices (blue)

Hyperparameter optimization was conducted in a brute force way of simply trying different combinations of hyperparameters and calculating RMSE and MAE values to determine the best hyperparameters. The tested hyperparameters are listed in Table 7.

Table 7. LSTM hyperparameters

Hyperparameter	Values
Epochs	20, 50
Neurons	10, 20, 40, 60, 80
Optimizers	Adam, RMSprop, SGD
Learning Rates	0.001, 0.0005, 0.0001
Batch Sizes	1, 4, 12, 24, 48, 168

The best hyperparameter combination was a model trained in 20 epochs with 80 neurons in the first LSTM layer optimized by Adam with a learning rate of 0.0005 and in batches of 24. The model had an RMSE of 6.18 and MAE of 3.61.

4.10.6 Encoder-decoder LSTM

To achieve the objective of the thesis, it is necessary for these models to forecast more than one hour ahead of the current time. The model needs to have a multistep output. This means that when the previous model was made to predict, the output

was a single float number, the output needs to be modified to be a sequence of numbers, that would be the forecast horizon.

The encoder-decoder LSTM contains two sub-models, the encoder and decoder. The encoder creates an internal representation of the input sequence, which the decoder will then use to predict a sequence. This is called a sequence-to-sequence model which has been used to automate translations between languages. (Brownlee 2018a)

The input and output timesteps that the model is trained on can be modified. This means that the model can take as input a week's worth of data or only a day's worth of data and forecast either the next day or the next week spot prices, for example. This was used to change the forecast horizon of the model.

The network architecture is visualized in Figure 41. The network has two hidden layers. It utilizes a repeat vector to repeat the input between two LSTM layers. The CuDNNLSTM is an optimized version of the regular LSTM layer, which can be only trained with Nvidia's graphics processing units.

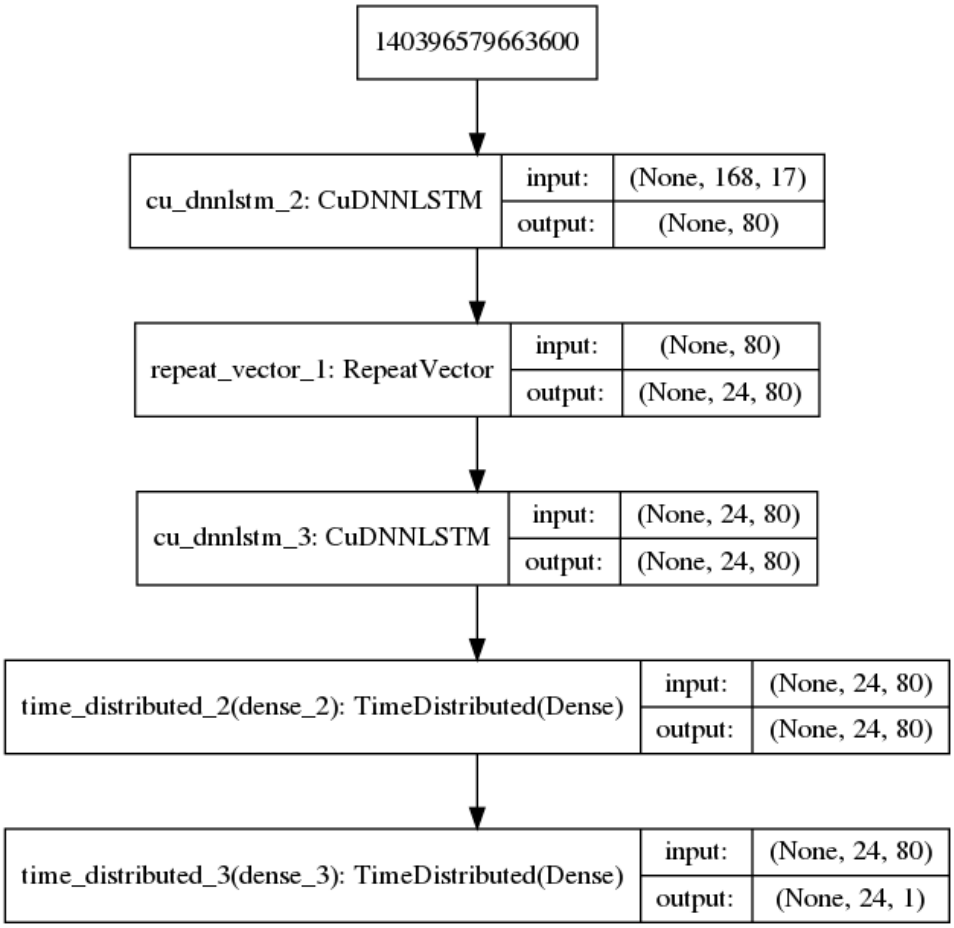

Figure 41. LSTM Encoder-decoder architecture

After some training and testing, a modification to the network was tested, where two more hidden LSTM layers were added, as visualized in Figure 42. The network training time increased slightly, but test results saw some improvement.

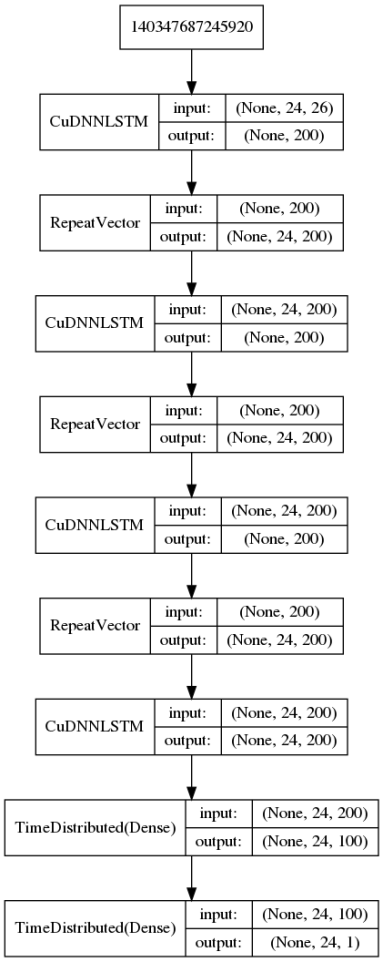

Figure 42. More depth added to the network

Input weight regularization and low learning rate of 0.00001 were also found to reduce loss and improve forecast accuracy. Learning rates of 0.01 or higher made the model ignore daily seasonality and draw flat lines across the forecast horizon as a result. This problem appeared also when the forecast horizon was a week or longer. The hyperparameters that achieved the best results are listed below in Table 8.

Table 8. Best LSTM Encoder-decoder training hyperparameters

Hyperparameter	Value
Optimizer	Adam
Learning rate	0.00001
cu_dnnlstm_2 and cu_dnnlstm_3 layer neuron amount	100
dense_2 layer neuron amount	100
dense_3 layer neuron amount	50
Epochs	50
Batch size	12

4.10.7 CNN-LSTM

The previous LSTM Encoder-decoder was changed to include convolution layers. This was achieved by adding two one-dimensional convolution layers and using one-dimensional max pooling, as visualized in Figure 43. The output from the convolutional layers is flattened and sent to the LSTM layer. This architecture is similar to the encoder-decoder, in this case the encoder is the CNN and the decoders is the LSTM. (Brownlee 2018a)

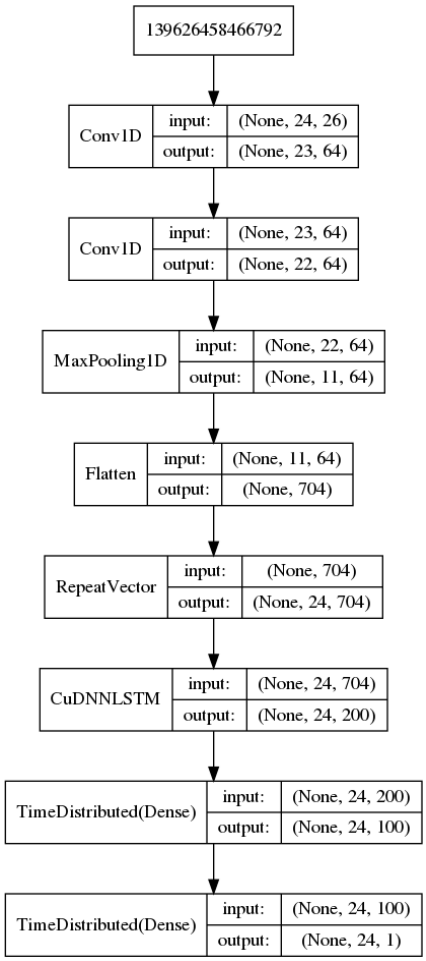

Figure 43. CNN-LSTM network architecture

The CNN-LSTM network was trained on the dataset, where the spot price was converted to a residual value by removing trend and seasonality. Better results compared to the previous datasets were not achieved using this method and converting the forecasts back to real values by adding back trend and seasonality to the forecast proved to be difficult and impractical. The training dataset was switched back to the dataset that was used to train the encoder-decoder model before. The forecasts compared to real spot prices are visualized below in Figure 44.

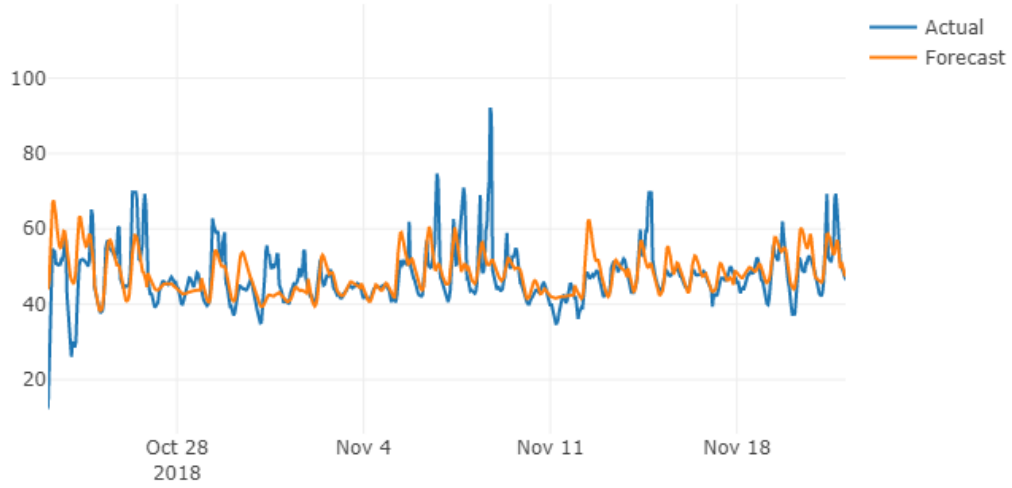

Figure 44. CNN-LSTM results with forecast horizon of 48

4.10.8 Ensemble learning

Ensemble learning was implemented by creating a stacking model, where predictions of multiple models were combined. mean, median and a neural network metalearner were tested as the combination methods. The best combination method was found to be the metalearner, where the neural network was a fully-connected neural network with two hidden layers.

The models to combine where selected from previous test results. For a 24-hour forecast, the best result was achieved by combining the forecasts of the LSTM network, autoregressive model and Prophet. A 36-hour forecast was also created to test how the performance compared to a 24-hour forecast. In the 36-hour ensemble model, three LSTM models were combined with autoregressive and prophet model forecasts. In 48-hour forecast, a combination of four LSTM models achieved the best results. Since the LSTM models performed poorly on 168 hours ahead forecasts, they

were not combined in the 168-hour ensemble model and instead autoregressive, Holt-Winters and Prophet models were used.

5 Results

The best forecasts were, depending on the forecast horizon, usually achieved with neural networks or the ensemble model, where the forecasts of models' autoregression, Prophet and multiple LSTMs were combined, and the forecasts were given weights with a separate metalearner neural network. After many rounds of feature engineering, the dataset that was constructed had the following features:

- Cyclical hour, day, weekday and month.
- Electricity consumption in Finland and Sweden
- Emission allowances price
- Hydro power production in Sweden
- Hydro reservoir levels in Norway and Sweden
- Nuclear power production in Finland and Sweden
- Precipitation near Harsprånget
- Prophet's daily, weekly and yearly seasonality
- Spot price, moving average and standard deviation from X hours ago, where X is the forecast horizon
- Temperature in Finland and Sweden
- Wind power production in Sweden

Best results from all the forecasting models were documented and written on the tables below. The lowest errors per error metric are bolded. The results from all the 24 hours ahead forecast models were written down on the Table 9 below. The best performing models were autoregressive, LSTM, CNN-LSTM and ensemble, where their error values were quite close to each other.

Table 9. 24 hours ahead forecast errors

Model Name	MAE	MAPE	RMSE
Baseline	5.34	10.28	9.06
AR	4.90	**9.04**	8.10
Holt-Winters	6.89	13.65	9.71
Prophet	6.84	9.15	13.99
LSTM	**4.80**	9.17	7.45
CNN-LSTM	5.40	9.98	7.99
Ensemble	4.96	9.41	**7.12**

Table 10 shows the results for models with forecast horizon 36. The encoder-decoder LSTM is the best model in this horizon on almost all metrics. The CNN-LSTM and ensemble were the second-best models and had almost similar error values.

Table 10. 36 hours ahead forecast errors

Model Name	MAE	MAPE	RMSE
Baseline	11.04	21.89	15.30
AR	6.39	11.88	9.53
Holt-Winters	9.14	17.90	12.00
Prophet	6.84	**9.15**	13.99
LSTM	**5.08**	9.57	**8.08**
CNN-LSTM	5.41	10.10	8.40
Ensemble	5.30	10.30	8.24

The results with forecast horizon 48 were recorded in Table 11 below. This time the CNN-LSTM model was the best based on MAE and RMSE metrics. The autoregressive, LSTM and ensemble models share the second place with mostly similar error values.

Table 11. 48 hours ahead forecast errors

Model Name	MAE	MAPE	RMSE
Baseline	7.47	14.59	11.74
AR	5.57	10.22	9.24
Holt-Winters	8.74	17.31	12.51
Prophet	6.84	**9.15**	13.99
LSTM	5.41	10.04	8.17
CNN-LSTM	**5.08**	9.76	**7.77**
Ensemble	5.44	10.25	8.14

The one week forecast horizon results were documented in Table 12. The neural networks performance was worse than what appears on the error metrics, since the prediction for the week is almost a straight line. The neural networks were excluded from the ensemble model for this reason. The autoregressive and ensemble model are close to each other in performance. However, the ensemble model is essentially the AR model enhanced with Prophet's predictions, so the ensemble model pulls ahead.

Table 12. 168 hours ahead forecast errors

Model Name	MAE	MAPE	RMSE
Baseline	7.02	13.83	11.24
AR	5.75	**10.65**	9.14
Holt-Winters	8.80	17.74	11.63
Prophet	6.84	13.99	9.15
LSTM	6.62	13.14	9.93
CNN-LSTM	7.31	13.58	10.48
Ensemble	**5.73**	11.01	**8.05**

To summarize, the best 24 hours ahead forecast was made by the LSTM model, where the forecast differs ± 4.80 euros from the spot price on average. The best 36 hours ahead forecast was performed by LSTM again, where the forecast error is ± 5.08 euros. The best 48 hours ahead was achieved by the CNN-LSTM model, where forecasts were ± 5.08 euros different from spot prices on average. The best one-week ahead forecasts were achieved by the ensemble model, where the forecasts were off by ± 5.08 euros on average.

A server programmed with Python's Flask library and JavaScript was used to visualize the results of forecasting models. When models were trained, the forecasts and real values were saved to a CSV file. The server lists the CSV files for the user and when a file is selected, the website creates a line graph of actual values and forecast values

with Plotly's JavaScript library. Plotly allows the user to zoom in the data and view the timestamps of all individual datapoints. Flask allows for easy usage of Jinja2 template engine to render HTML templates.

The server also has a data analysis feature, where the features used in training the machine learning model can be drawn into a graph. Due to the high scale differences between some of the features, all the features are min-max scaled before they are drawn into the graph.

6 Discussion

Learning a proper machine learning workflow was the first challenge. The testing framework, naming convention for models and recording hyperparameters should have been done before the neural network training was started. Instead these were implemented as the need for them arose, which caused unnecessary information loss and confusion during research. Otherwise a great deal was learned about handling time series data, different forecasting methods, using statistical models and training machine learning models.

Finding good features that improve model performance was the hardest part of the research. The neural networks take a long time to train, and since they are stochastic by nature, the training results will not be the same even with identical hyperparameters. Neural network architecture and hyperparameters affect the accuracy a great deal. These factors made finding a good data group hard, thus research was put into reading about spot price drivers from publications, then downloading datasets from open data sources and using them as features.

Neural network models were unable to react to sudden price spikes in the spot price. This is understandable, because the price spikes may happen for a variety of reasons, and the best course of action would be to design a separate model to predict price spikes themselves. This price spike prediction model could then be combined with

current models to improve performance. Further feature engineering and optimizing the used neural networks or switching to a new architecture could improve model accuracy and lower forecast error. When using sequence that were the length of a week or longer, the neural networks predictions would flatten to an almost even line. This happened because the sequence was too long and instead of fluctuating daily, the neural network thinks the best way to reduce loss is to just draw a flat line. This could be fixed by allowing length of input sequence and output sequence to be different.

The server application, Jupyter notebooks and Python scripts used for data analysis, data manipulation, training models and forecasting are available from Gitlab at https://gitlab.labranet.jamk.fi/korpjo/nordpool-spot-price.

References

Box, G., Jenkins, G., Reinsel, G. & Ljung, G. 2015. Time Series Analysis : Forecasting and Control. Hoboken: Wiley.

Brink, H., Richards, J. & Fetherolf M. 2017. Real-world Machine Learning. New York: Manning Publications Co.

Brownlee, J. 2016. What Is Time Series Forecasting? Accessed on 10 March 2019. Retrieved from https://machinelearningmastery.com/time-series-forecasting/

Brownlee, J. 2017. Autoregression Models for Time Series Forecasting With Python. Accessed on 3 April 2019. Retrieved from https://machinelearningmastery.com/autoregression-models-time-series-forecasting-python/

Brownlee, J. 2018a. How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption. Accessed on 22 March 2019. Retrieved from https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/

Brownlee, J. 2018b. Ensemble Learning Methods for Deep Learning Neural Networks. Accessed on 16 April 2019. Retrieved from https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/

Buduma, N. & Locascio, N. 2017. Fundamentals of Deep Learning. Sebastopol: O'Reilly Media.

Cady, F. 2017. The Data Science Handbook. Hoboken: Wiley

Donahue, J., Hendricks, L., Rohrbach, M., Venugopalan, S., Guadarrama, S., Saenko, K. & Darrell, T. 2016. Long-term Recurrent Convolutional Networks for Visual Recognition and Description. Accessed on 3 April 2019. Retrieved from https://arxiv.org/pdf/1411.4389.pdf

European Commission. 2015. EU ETS Handbook. Accessed 16 April 2019. Retrieved from https://ec.europa.eu/clima/sites/clima/files/docs/ets_handbook_en.pdf

European Network of Transmission System Operators for Electricity. 2018. Statistical Factsheet 2017. Accessed on 12 April 2019. Retrieved from https://docstore.entsoe.eu/Documents/Publications/Statistics/Factsheet/entsoe_sfs_2017.pdf

Hyndman, R. & Athanasopoulos, G. 2018. Forecasting: principles and practice, second edition. Melbourne: OTexts.

Leskinen, J. 2018. Sähkön hinnassa kova nousu – 58% kalliimpaa kuin viime kesänä. Accessed on 25 February 2019. Retrieved from

https://www.uusisuomi.fi/raha/255395-sahkon-hinnassa-kova-nousu-58-kalliimpaa-kuin-viime-kesana

London, I. 2016. Encoding cyclical continuous features – 24-hour time. Accessed on 11 March 2019. Retrieved from https://ianlondon.github.io/blog/encoding-cyclical-features-24hour-time/

Montgomery, D., Jennings, C. & Kulahci, M. 2015. Introduction to Time Series Analysis and Forecasting. Hoboken: Wiley.

Morison, R. & Hodges, J. 2018. Carbon Reaches 10-Year High, Pushing Up European Power Prices. Accessed on 16 April 2019. Retrieved from https://www.bloomberg.com/news/articles/2018-08-23/carbon-reaching-20-euros-a-ton-in-europe-raises-price-for-power

NIST/SEMATECH. 2012. e-Handbook of Statistical Methods. Accessed on 15 April 2019. Retrieved from https://www.itl.nist.gov/div898/handbook/

Nord Pool. N.d. Day-ahead trading order types. Accessed on 1 February 2019. Retrieved from https://www.nordpoolgroup.com/trading/Day-ahead-trading/Order-types/

Nord Pool. N.d. History. Accessed on 1 February 2019. Retrieved from https://www.nordpoolgroup.com/About-us/History/

Nord Pool. N.d. Price calculation. Accessed on 7 February 2019. Retrieved from https://www.nordpoolgroup.com/trading/Day-ahead-trading/Price-calculation/

Parr, T., Turgutlu, K., Csiszar, C. & Howard, J. Beware Default Random Forest Importances. Accessed on 25 February2019. Retrieved from https://explained.ai/rf-importance/index.html

Peljo, J. 2013. Futures Pricing in the Nordic Electricity Market. Master's thesis. Aalto University, Master of Science, Degree programme in finance. Accessed on 8 March 2019. Retrieved from http://epub.lib.aalto.fi/en/ethesis/pdf/13203/hse_ethesis_13203.pdf

Raschka, S. 2014. About Feature Scaling and Normalization. Accessed on 26 April 2019. Retrieved from http://sebastianraschka.com/Articles/2014_about_feature_scaling.html#standardization-and-min-max-scaling

Scikit-learn. N.d. Sklearn.preprocessing.MinMaxScaler. Accessed on 5 March 2019. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

Sebastian-Coleman, L. 2012. Measuring Data Quality for Ongoing Improvement : A Data Quality Assessment Framework. Waltham: Morgan Kaufmann

Statistics Finland. 2018. Energy Consumption in households 2017. Accessed on 18 April 2019. Retrieved from https://www.stat.fi/til/asen/2017/asen_2017_2018-11-22_en.pdf

Statistics Finland. 2018. Electricity consumption by Year(*preliminary), Sector and Data. Accessed on 19 April 2019. Retrieved from http://pxnet2.stat.fi/PXWeb/pxweb/en/StatFin/StatFin__ene__ehk/statfin_ehk_pxt_013_en.px/

Taylor, S. & Letham, B. 2017. Forecasting at Scale. Accessed on 30 March 2019. Retrieved from https://peerj.com/preprints/3190.pdf

Van der Merwe, R. 2018. Implementing Facebook Prophet efficiently. Accessed on 27 March 2019. Retrieved from https://towardsdatascience.com/implementing-facebook-prophet-efficiently-c241305405a3

Vasilev, I., Slater, D., Spacagna, G., Roelants, P. & Zocca, V. 2019. Python Deep Learning Second Edition. Birmingham : Packt

Voronin, S. 2013. Price spike forecasting in a competitive day-ahead energy market. Master's Thesis Lappeenranta University of Technology. Accessed on 25 March 2019. Retrieved from http://lutpub.lut.fi/bitstream/handle/10024/93793/isbn9789522654625.pdf?sequence=2

Willmott, C. & Matsuura, K. 2005. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. Accessed on 5 April 2019. Retrieved from https://www.int-res.com/articles/cr2005/30/c030p079.pdf