Department of Mathematics
and Computer Science

University of Southern Denmark

Master Thesis in Data Science

Exploring climate data's relevance to predict Energy consumption

Why is energy capitalized?..

Author
Mathias Østergaard Hansen
Exam nr: 453228
mathh17@student.sdu.dk

Supervisors
Vaidotas Characiejus(SDU)
Jeppe Miki Busk Olsen(Energinet)

May 23, 2022

SDU

Table of Contents

I would suggest to reduce the margins (they are way too large) and increase the font size.

Chapter 1

Introduction

In today's society where countries are shifting from fossil fuels towards renewable energy, it becomes increasingly important for these countries to be able to sustain a reliable electrical grid. If more and more sectors become dependent on renewable energies, which in Denmark consists of mostly wind and sun electricity, the consequences of the electrical grid going down becomes worse. A big part of sustaining a reliable electrical grid is being able to allocate the need for the electricity and supply in advance. The transmission system operators(TSOs) handles this responsibility, and a big part of this is forecasting the electricity consumption in advance. The more precise these forecasts are, the better these TSO's can plan the production and behavior of the needed electricity. The topic of this thesis is to investigate the use of climate data in the predictions and forecasts of electricity consumption. The reason this could be an interesting topic is because in Denmark approximately 80% of the produced electricity is used for heating homes, Energistyrelsen 2019. By these figures, it would be interesting to explore if the use of weather data would have a beneficial part when predicting the electrical consumption. This project will be carried out in collaboration with the danish TSO, Enerignet, in order to gain initial insights in their data, experiences and needs for such forecasts. In Denmark the Electrical grid can be divided in two zones, DK1 and DK2, where DK1 is the west part of Denmark from Storebælt and DK2 is the east part. Energinet is interested in seeing the performance of machine learning models for the such forecasts on electricity for these two zones. Furthermore, more specifically are Energinet interested in hourly forecasts, which will the main focus of this project.

This should be a separate paragraph.

Chapter 2

Data Collection and Preprocessing

1 Observational data from DMI

In 2020 DMI began uploading complete data sets of their weather data through an API-service. There are four data sets available through their API, as of yet, the two that could be relevant to the project would be "climate data" and "Meteorological Observation", metObs. Climate data is the latest release for the API and contains very little data, there are a lot of holes in the data set and very few stations with data on for relevant parameters. The good thing about the climate data set is that is quality checked and rinsed for outliers.

The metObs data set is a dataset of a wide range of meteorological observations recorded by DMI, all the way back to 1953 actually. The data set is complete with very few gaps, but the downside to this dataset is that the data set is not quality checked and requires the user to handle erroneous data. There are ways to deal with outliers and in the context of this project it is easier to deal with outliers rather than dealing with a mostly incomplete data set, as the climate data set is still very incomplete. DMI records the weather data from its weather stations placed around all of Denmark. DMI describes the occurrence of outliers usually comes from malfunctioning sensors, wear and tear, and extremely rarely vandalism.

In the case of this project the two interesting parameters from the metObs data set are the temperature mean and radiation mean. These two variables are means over an hour of data. Which means the possible outliers will be watered down before being reported. In this project, each recorded observation will be added together with a set of observations from other weather stations and then have another mean calculated from those values. Thus impact of an outlier from one reading will be very small. Therefore, the only action that will be taken against outliers for this dataset is the looking over the data sets for possible malfunctioning stations and removing them. Which ended up being two stations that would continuously report wrong data for radiation.

Later on when the Climate dataset is fully developed this would be the preferred data set.

When the data set is downloaded through the API, the data is structured station by station, then each station contains a dataframe of all the recorded observations. For the purpose of predicting the consumption for the areas DK1 and DK2 the weather stations will be divided into three groups, DK1,DK2 and neither. The stations that get placed in the neither category will be excluded. The criteria for excluding a station is if the station is placed either at sea or on the island of Bornholm. The reason for this is that Bornholm is only responsible for a minimal amount of the overall consumption of DK2 which is the area it would placed in, and the weather on Bornholm might differ significantly from the rest of DK2.

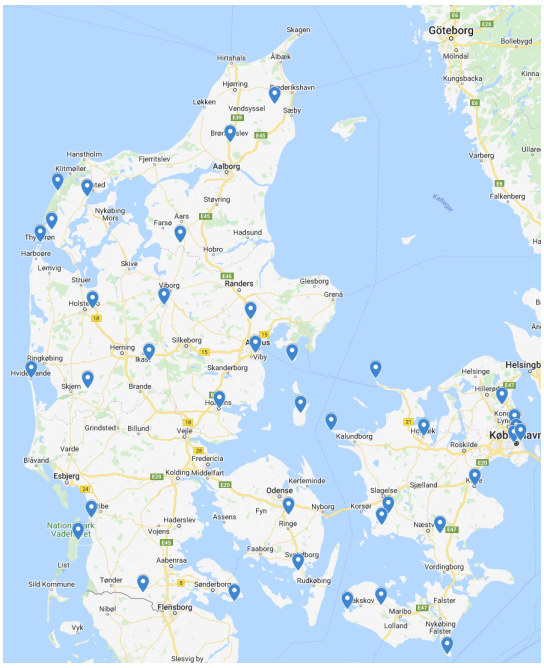


Fig. 1: Map of weather stations in the groups DK1 and DK2

This was done by taking the most eastern, western, southern and northern coordinates from both DK1 and DK2. The returned stations are then checked manually to see if it is in within the right category, and if it is not the data from the station is removed. This resulted in complete data from 25 stations for DK1, and 15 DK2, Figure 1. All the data from each station in each area are then added together to get a mean for each feature for the given price area. This will the data that is used to train the models. The reason for choos-

ing the features time, temperature and radiation comes from advice and experience from Energinet. From their experience it would be the two interesting features from the observations. Time and temperature is used given the amount of electricity used for heating. Radiation is added as a feature because Energinet wants to exclude the amount that residents produced from solar panels. Therefore an initial assumption is that there is a correlation between high radiation and a decrease in consumption. The solar panels produce electricity that is added directly to the danish electrical grid. This means Energinet will have to produce less electricity from the power plants because the solar produced electricity is not consumed by the single consumer but it is fed into the general grid.

2 Electricity Consumption Data from Energinet

Just like DMI, Energinet's data is available through an API. From their web portal: energidataservice.dk. The relevant data set for this project is the one called *"Production and Consumption - Settlement"*. This is a data set of the over all electricity consumption divided into the two price zones DK1 and DK2, and fortunately this data set is quality checked and 99.9% correct. This data set contains many features, of which the price area, time, gross consumption and solar power self consumption will be the ones relevant in this case. The feature, solar power self consumption, will not be a part of the final data set, but the data from this feature will be deducted from the gross consumption feature. The reason for this is because Energinet wants to know how much electricity they need the power plants to produce, and that is exclusive of this production from the solar panels.

3 Meteorological Forecast Data

When the models have been trained on the observed data from the MetObs dataset, the models should be able to use similar forecast data from DMI. Energinet receives forecast data from DMI in many different time resolutions, for this project the data set will be the hourly forecast data which comes in every 3rd hour 54 hours ahead. The way this is structured could be expressed as such: $\hat{T}_{24(k-1)+j,h}$. Where k represents the day of the year in a range from 1 to 365, j represents the hour of that day in a range from 1 to 24 and h represents

I would suggest to introduce notation for all data in the thesis.

the number of hours ahead the values were predicted. As an example, the temperature 20 hours ahead for the date 5/1 at 8 a.m. would be represented as $\hat{T}_{24(5-1)+8,20}$ This allows for predictions being done for single time steps 48 hours ahead, for each available time step. For the purpose of this project, the time horizon that the models will try to make forecasts for is the year of 2019. Thus the model will be tested on a complete seasonal year, and it should yield enough predictions to give insights to the performance of the models. An interesting topic to explore was how precise the forecast data were in relation to the actual observed values. Also, how much does the time ahead the values is predicted affect the precision of the forecast?

This jumps a bit out of context.

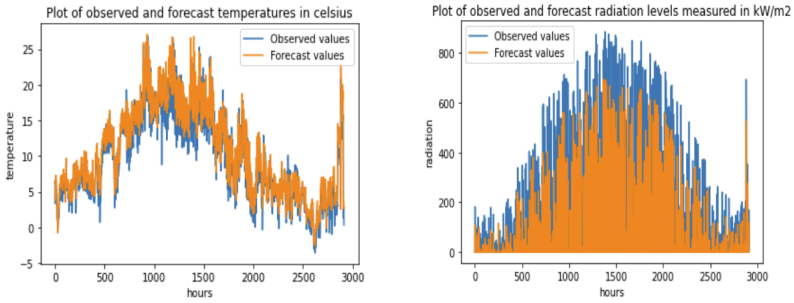


Fig. 2: plot of observed and forecast data for the temperature and radiation for forecasts 48 hours ahead only.

From Figure 2, the forecasts for temperature seems to fit the observed values rather well. However, the forecast radiation values seems to deviate significantly from the observed values. looking at the plot it seems to mostly underestimate the radiation. In order to get an understanding of how precise these forecasts are the R^2 value was calculated for both 1 and 48 hour ahead forecasts.

R^2 values for forecast data		
	1 hour ahead	48 hours ahead
temperature	94.2	93.8
Radiation	70.5	67.4

Table 1: R^2 values for the year 2019 for 1 and 48 hours ahead forecasts.

Looking at the R^2 values from table **??** for temperature, DMI is able to adequately forecast the temperature over the course of a year. Both for 1 and 48 hours ahead, the reduction in the R^2 value going from 1 hour to 48 hours is minimal and does not appear to be of any concern. However, the R^2 values for radiation suggests the same as the plots. There are significant errors in the forecast of radiation, when compared to temperature. This does make sense,

since when it comes to radiation, the value is much more locally variable than temperature. The reason for this is that cloud coverage has quite the effect on radiation and it is difficult to predict the precise placement of the clouds. In terms of data quality this might turn out to be quite the factor for eventual error in the predictions, since you want the training data to be as similar to the actual data as possible. It should be investigated if this feature should remain in the final dataset, since it might end up affecting the precision of the model.

Chapter 3

Data Preprocessing and Exploration

Developing machine learning models require a deep understanding of the available data. In order to get a overview of the correlations and relations between the data from DMI and Energinet the data will be explored thoroughly and visualized. There are some initial assumptions about the data that will be investigated during this process:

These are not assumptions. You could formulate them as hypotheses but I would suggest to formulate them as the results of exploratory data analysis.

- There is a weekly pattern of power consumption where there is a decline during weekends and holidays.
- There is a correlation between the time of day and consumption.
- There is a negative correlation between sun radiation and consumption.
- There is a correlation between temperature and consumption.

These assumptions will be either confirmed or denied through the exploration of the data, or literature from relevant studies.
Firstly the time of the observations will have to be organized in a manner that allows to fully investigate it's impact on consumption. For the purpose of this project where there are behavioural patterns for electricity during the week and holidays, a binary feature will be added to the data set, if it is a holiday or not. This feature should capture the decline in consumption that evidently happens on weekends and holidays Ziel 2018.

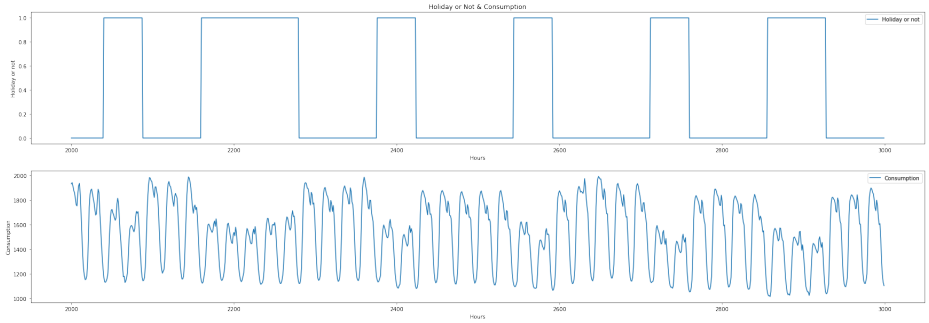


Fig. 3: Graphs over consumption and holiday feature over a period of time, specifically around Easter

Looking at the behaviour of consumption and weekends or holidays it appears that there is a correlation between weekends and consumption, Figure 3. The consumption appears to decrease during the weekends, which confirms the results put forth by Ziel. This pattern possibly comes from a large part of the industrial sector closes down over weekends and public holidays. Ziel also compares the effect of public holidays like easter over a seven year period, and it appears that the impact of holidays is rather stable. This representations of days of the week seems to capture the behaviour of the electricity consumption in a sufficient manner.
Since most of the danish population is sleeping during the night it is expected that the electricity consumption is reduced during those hours, but exactly how is the behavioural pattern of the consumption in relation to the hour of the day?

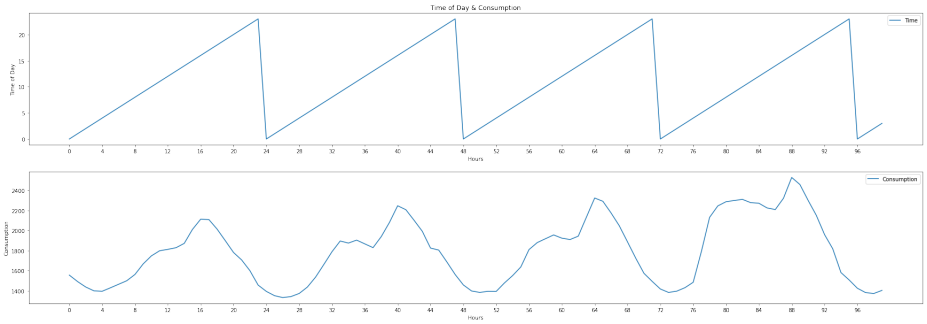


Fig. 4: Graphs over consumption and time of the day over a period of time

As expected there is a clear correlation between the time of day and electricity consumption. Something that seems apparent from the graph, Figure 4, is that there are daily peaks after 4pm, where people return from work. This makes sense since people generally will be preparing dinner, and using their appliances to wash their clothes etc.
During the last 10 years solar panels have become increasingly more popular in Denmark, in September 2019 106,461 soloar panels to be exact, Bolius. As stated earlier, this is not a form of consumption that Energinet needs to predict, therefore the consumption is deducted from the gross consumption. By now, with so many solar panels installed the effect from these solar panels is assumed be noticeable on the gross consumption.

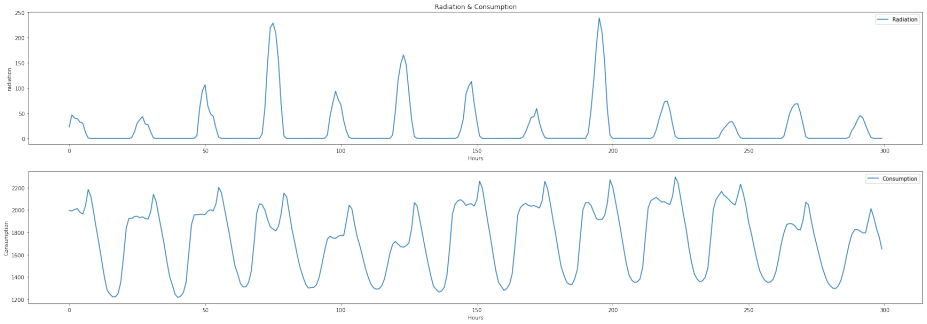


Fig. 5: Graphs over consumption and radiation over a period of time

Inspecting the graph, Figure 5, it is possible to see a noticeable dip in consumption on the days with high radiation. Around 75 hours, and 200 there are small dips in the consumption, which could be caused by the increased radiation. There are also other possible reasons this could happen. It might be because people are more willing to spend time outside when the sun is shining, and that they just simply are not using their appliances.

Since such a big amount of the residential electricity consumption is used for heating up houses, it is expected to see some form of correlation between temperature and electricity. In the danish heating industry, the companies convert the temperature to degree-days to best capture the energy required to heat up a house, *Energihåndbogen* 2019. Degree-days is a unit for measuring the difference between the mean outside temperature and the mean inside temperature, in this case being, 17 degrees Celsius. This gives a degree-day of 5 if the outside temperature is 12°and -2 if the outside temperature is 19°. It can also be expressed as $degreedays = -C + 17$, where C is the temperature in Celsius. The reason the heating industry uses this unit rather than just using temperature, is because this best captures the impact the cold has on the energy required to keep a house at the mean temperature. By advice from Energinet, this will also be the unit used for this project, since Energinet also uses degree-days when measuring temperature in relation to energy consumption.

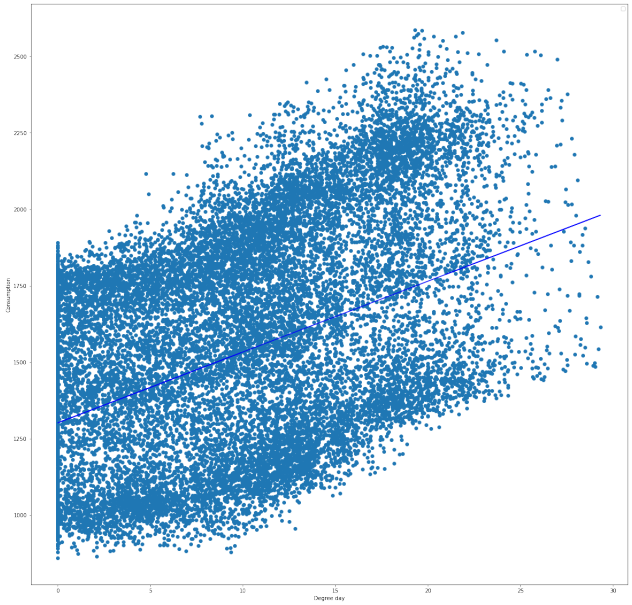


Fig. 6: Plot of the consumption in relation to degree-days with a tendency line

There appears to be a correlation between degree-days and the electricity consumption, Figure 6. As the degree-days gets closer to zero it does seem like the correlation begins to stagnate, which fits the narrative stated earlier, that after a certain point the temperature has no effect on the consumption.

4 Feature importance

After having visually inspected all the relevant features available, a final step was taken to see the importance of these features. As it is visible from the graphs and plots above, the correlations are not equal. Therefore, to get a better understanding of the information a machine learning model is able to extract from these features, the SHAP values from the features will be calculated. By calculation a predictions SHAP values, SHapley Additive exPlanations, you can break down the impact the values of each feature has on the prediction. This is extremely useful for checking for redundant features in your data set. Shapley values are based on a solution concept from game theory used to determine the contribution of each player, and the name shapley comes from the mathematician, Lloyd Shapley who introduced the theory in 1951. In 2017 this theory was used in machine learning, Scott M. Lundberg 2017, with a python package that calculates these values based on ones pre-

dicter, SHAP. In order to calculate the SHAP values a very basic predicter will be implemented, in this case a XGBoost predicter. XGBoost is a boosting algorithm which boosts many small decision tree, for calculation the SHAP values on a prediction like this, the "tree interpreter" function will be used from the SHAP package. Using the simple XGBoost model the SHAP values calculated for the features can be presented in a summary plot:

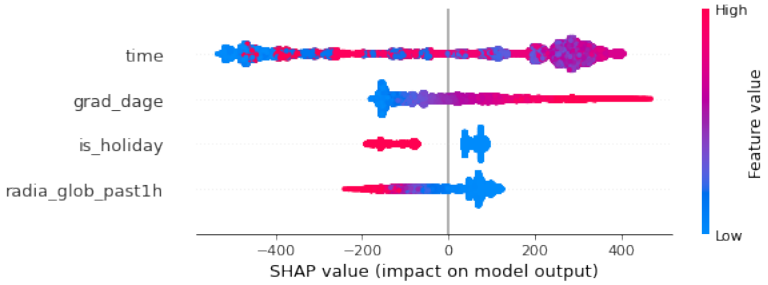


Fig. 7: Summary plot of the SHAP values, for each feature

Fig. 8: Plot of the SHAP values for temperature

Degree-days have a nice gradient coloring throughout the whole plot. It seems to be very clear to the model that a high value degree-day means an increase in consumption. the impact of appears to be more consistent for degree-days than the time feature. The degree-day feature stops abruptly for low values, this is at the 17 degrees, where the degree-days are capped and goes to zero. In order to check whether or not the model is able to gather some information from higher temperatures i made the same plot, but with temperature in Celsius, Figure 8, and it appears that the temperature has the same impact as degree-days. For the is_holiday feature all values that indicates a weekend or a holiday seem to produce a negative deviation for the consumption, which is in line with the correlation found in the plot.

The radiation plot shows that the impact of the radiation actually causes a negative deviation, and this confirms that the model is actually able to pick up on the small dips that appeared to be present on sunny days. Since the temperature stops yielding information for the model when the temperature is above 17 degrees, hopefully this feature is able to give some information on sunny days during the summer, where the model does not get information from degree-days. However radiation appears to be the least important

feature, and since the forecast values for radiation is not more precise, the performance of the models should be investigated excluding the radiation feature. However, the time feature seems to be more difficult to decipher. Since the SHAP values represents data in a continuous manner, representing the time feature is not very good, since it is cyclical and not continuous. This means whenever the time passes midnight, the feature value goes from the highest possible value to the lowest, but the consumption is not cyclical. This explains the mix of the peak values throughout the plot. Looking at the right side of the plot it would appear that the model is able to pick up the daily peaks in the data, that happens around 16 when people return home from work.
After inspecting both the data correlations and their impact on data models, there does not seem to be any redundant features. The final data set that will be used for training looks like this

Training Dataset				
Time	Degree-days	Radiation	holiday	Consumption in MWh

Table 2: Features in the data set that will be used for training the model

Chapter 4

Literature Review

Predicting energy consumption is not a new subject of study in the field of machine learning. There are wast amounts of papers, reports and projects on the subject. With the increase in popularity for the subject of machine learning and especially neural networks in the last 20 years, many new methods of predicting this consumption have been developed. In the report Iman Ghalehkhondabi 2016 it is made clear how big of an increase forecasting energy consumption has seen, and that is just from 2005-2015. In the figure from the report, the increase is depicted in a diagram showing the growth from 1994-2014, which appears to be exponential. This report indicates that there is a big interest in the subject. In the report they state that the reason for the importance of being able to predict the electricity consumption has many factors. Distribution of electricity is a big factor, since you do not want to overproduce, or under produce. In Denmark overproducing is handled by exporting excess power to neighbouring countries. This happens quite frequently because Denmark is able to produce large amounts of electricity by wind. Therefore correctly distributing and allocation of electricity is of big importance. To get some perspective on state of the art models and what is considered best practice on this topic, relevant literature will have to be researched and reviewed. Review papers will be the main way of gaining a broad perspective on relevant models and implementations, given the wast amounts of research on the topic. In the report, Antón Román-Portabales 2021, a wide variety of papers on the subject of Artificial Nerual Networks(ANN's) used to forecast electricity demand are reviewed. In total they review 56 papers, and interestingly enough 46 of the papers are on Short Term Load Forecasting(STLF). It appears from the paper that the most popular load forecasting method is short term, in terms of ANN's. From the applicability of ANN's and the way they use data to make predictions this makes sense. Because ANN's is very reliant on the quality and precision of the data available to the model, so making long term load forecasting(LTLF) or even medium term would seem difficult. STLF seems to be predicted from weather forecasts, which becomes more and more

incorrect the further in the future the forecasts are done. In terms of best performing models the paper states that usually the best performing models are models that are able to capture recurrent patters, such as recurrent neural networks(RNN). Which is in line with they claim that using the previous energy consumption for forecasting in the future is a better parameter than the weather. Especially the Long Short Term Memory,LSTM, is able to achieve some very good results. An interesting note is that the paper states that with the LSTM the predictions tend to become more accurate the more consumers the data is aggregated over, which might be of interest for TSO's such as Energinet, if they want to implement forecasting in smaller scales than the zones DK1 and DK2. The review paper, Jason Runge 2021, comes to similar conclusions as the previous paper when applied to larger scale predictions. However in one of the papers that is reviewed, the extreme gradient boosting algorithm, XGBoost, shows the best results. But in that specific paper it seems to be a regular Deep Neural Network (DNN) rather than a form of RNN, that the XGBoost outperforms. In the review paper, Hristos Tyralis 2021, they state the use of boosting algorithms as being "underexploited". The modern boosting algorithms, such as XGBoost, LightGBM and CatBoost, have shown state-of-the-art prediction performances. The paper goes into great length to explain the ease of implementation of these modern boosting algorithms. Especially the XGBoost algorithm is generally perceived as having a very good out-of-the-box performance. This is in line with one of the conclusions from Iman Ghalehkhondabi 2016 in which they state:*"As a conclusion, developing simple methods with acceptable accuracy is an attractive area of future studies."* Since the computational resources needed for these very deep neural networks are significantly higher than the newer machine learning algorithms, such as gradient boosting algorithms. From the review papers, the state-of-the-art methods of load forecasting seems to be using a form of RNN, more specifically LSTM's seems to be very popular. Boosting algorithms also seem to be able to achieve acceptable performances. These two types of models will be the ones that will be implemented in this project. The LSTM is chosen because it seems to be the most popular for this specific task, and XGBoost is chosen because from the review papers they show great promise.

Chapter 5

Methodology

Now that a general understanding of what is perceived to be some of the best practices for modeling load forecasting, it is necessary to understand the chosen methods on a deeper level. In this section the models: Gradient Boosting, specifically XGBoost and the RNN model LSTM will be explained in order to gain an in-depth understanding of the models.

5 Extreme Gradient Boosting

5.1 Gradient Boosting

The general idea behind machine learning models is to construct one model that is able to make predictions. However, boosting algorithms aims to train a sequence of weak learners that as a whole is able to make predictions, also called ensemble learning. Each model is trained to compensate for the weaknesses of the model before. In the relevance of gradient boosting, the weak learners consists of small decision trees, which are sequentially improved upon based on the residual errors of the previous decision tree. The first tree in the sequence is usually leaf which predicts the average of the target values, for regression problems, and after that the boosting algorithm will gradiently reduce the error by adding more trees. Decision trees consists of a certain amount of binary statements, also called splits, these splits is used to either make a prediction or there will be a new split until they reach a prediction, Figure 9. Gradient boosting algorithms utilizes these trees as the weak learners they train sequentially.

The sequential connection of these trees means for each tree, that tree is fitted to the residual errors of the previous tree, which it then tries to minimize. In the case of regressional implementations of gradient boosting algorithms, the residual error is calculated by a loss function.

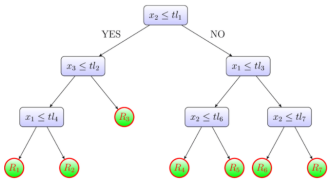


Fig. 9: Example of decision tree

There are supervised learning algorithms where making the residuals equal to 0 is not even possible.

Since gradient boosting is a supervised learning algorithm, it would be easy for these trees to fit itself to the training data and thus very quickly reduce the residual errors to zero. In order for this not to happen, each trees contribution is scaled by the learning rate. The learning rate ensures that each tree contributes the same amount. Friedman who was the one to first implement gradient boosting states that lower values for the learning rate typically yields better results Friedman 1999. There is however, a trade-off in terms of implementations. Having a low learning rate require more trees in the model and that increases the computational requirements for the model. Each tree then contributes to the overall prediction with a weight that is added to the rest of the trees weights. This means the model aggregates the weights of all the small trees, this value is then the deviance from the average value that the prediction is.

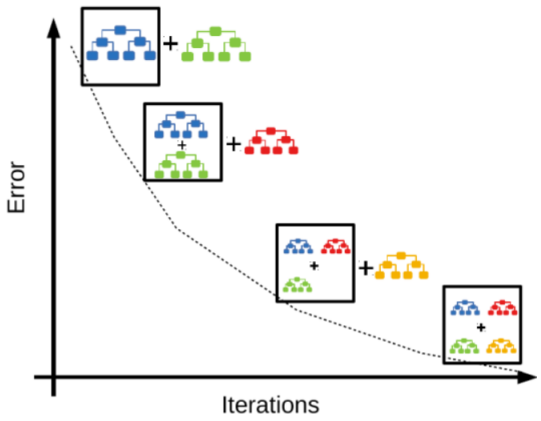


Fig. 10: Illustration of how the gradient boosting algorithm sequentially adds trees to reduce error

When implementing gradient boosting algorithms the depth of the trees will have to specified as it is one of the hyperparameters for this form of boosting algorithm. The trees calculated by gradient boosting algorithms have a fixed depth, and allows for the individual trees to be quite deep. However increasing the depth of the trees, also increases the computational recourses required by the model quite significantly.

5.2 XGBoosting

The objective of the XGBoost algorithm is to minimize the value of $\mathcal{L}$ in and additive manner given by the regularized formula:

Is this defined in the thesis?

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{1}$$

$\mathbf x_i$ is used in Chen and Guestrin (2016).

$$\text{Where } \Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2$$

What is T? It is only defined below Equation (2).

where l is the loss function that calculates the residual errors in that particular iteration step. f_t represents a tree structure q, and leaf weights w. γ and λ are the regularization parameters of the XGBoost model is a key difference in regular gradient boosting algorithms and the XGBoost algorithm. γ is the threshold for the pruning done on leafs, and λ aims to reduce the predictions sensitivity to individual observations, thus helping the model from overfitting. The initial prediction of the XGBoost algorithm consists of just the average value of the target value. From this, the residuals g, is calculated and is used to calculate the score of the structure in the tree using the following formula:

What is q?

Is q a tree structure or the quality of that tree structure?.. And what does quality of a tree structure mean?

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{2}$$

Where t is the iteration, q is the quality of that tree structure, T is the number of leaves in the tree, γ and λ are regularization terms. In XGBoosting the splits for the decision trees are based on a value $\mathcal{L}_{split}$. This is a representation of the information gained from performing the split on the specific values in the particular tree. The information gain of a split is then calculated by:

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{3}$$

Not the right way to cite the paper.

From the original paper for XGBoost,Tianqi Chen 2016, the computational algorithm for split finding is expressed as:

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Algorithm 1, the exact greedy algorithm, iterates over all possible splits and thus comes to the best possible split. But from the XGBoost documentation it is advised to only use this algorithm for smaller dataset since it becomes quite computationally heavy. Therefore the approximate algorithm was developed to increase XGBoost scaling capabilities for bigger data sets. In the paper "big dataset" are demonstrated as datasets with millions of rows, which is not the case here. Luckily, In the implementation that is done by XGBoost the model is able to decide which algorithm to use in the specific situation. Even though this helps reducing the computational requirements for finding the best split. XGBoost goes to an even greater extend by implementing column blocks for parallel learning. This lets the algorithm split up the dataset so multiple threads can store and work on calculating statistical properties. In the paper describing XGBoost, Tianqi Chen 2016, they state that at larger data sets the cache-aware algorithm is able to run twice as fast as the naive.

6 Long Short Term Memory

LSTM's are a form of Neural Network, NN, in the subcategory of NN's called Recurrent Neural Networks, RNN. It is therefor crucial to understand both NN's and RNN's to fully utilize the power of LSTM's.

6.1 Neural Networks

Neural networks are a set of algorithms that together tries to generalize over a set of training data enough to make predictions for a given feature. A NN in its simplest form consist of 3 types of layers, a input layer which contains the input data for the algorithms to generalize over. A Output layer which will contain a representation of the target feature in a value between 0 and 1. In the middle of these two layers are what is called the hidden layers. The reason the middle layers are called hidden layers is because the value these layers contain are usually not relevant for the user to know.

How is the predicted value converted to the actual consumption?

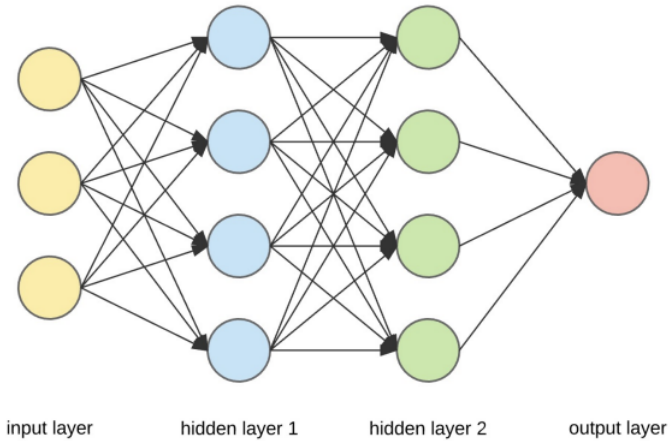


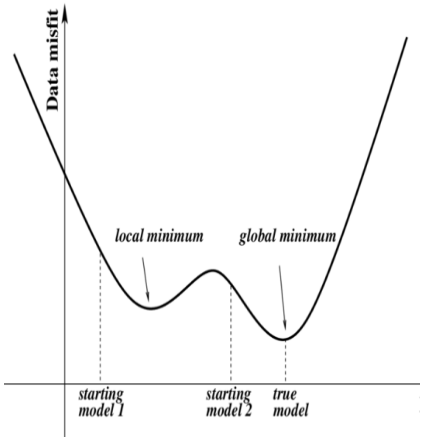
Fig. 11: Diagram of a simple neural network

In a implementation of a NN, there can be many hidden layers, but usually there will be one input and one output layer. The job for these hidden layers are to capture particular patterns for the data. Each node in the layers will try and capture some part of information from the data sent from the previous layer. The change in the information between the layers happens because each node is connected to the nodes in the previous layer, and these connections contains weights and biases. When we talk about training a neural network or that the network is learning, what is meant is that these weights and biases are tweaked and tuned to better capture the input data for the right output.

6.2 Gradient Descent

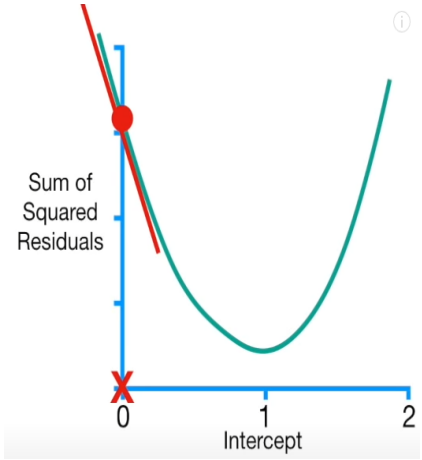
When a model's weights and biases are updated it is done through backpropagation. What is meant by updating the weights and biases for a model is

that they are updated in an effort to reduce the Sum of the Squared Residuals(SSR). Reaching the lowest point of SSR's is called reaching the global minimum, this is where the model is as precise as it can possible be. However, finding the global minimum cannot be guaranteed, the reason for this is the way the SSR are minimized during training. In order to decrease the SSR in a timely manner, the SSR is minimized by gradiently calculating the descent of the slope of the SSR. This is called gradient descent and is the way the SSR is minimized so that when the model updates the weights and biases the new values should reduce the SSR.



(a) Depiction of local and global minimum

Where does this diagram come from? A reference should be given unless it is generated by you.



(b) Depiction of gradient descent slope and the steps along the X-axis

Fig. 12: Caption for this figure with two images

This might seem simple for the model, the issue is that there might be multiple local minimum and the model might not be able to move past local minimum. With the initialization of the model being random this is the reason there is no guarantee for finding the global minimum. But with gradient descent the model will usually find a local minimum.

6.3 Recurrent Neural Networks

A problem that regular feed-forward neural networks suffers from are the lack of memory between predictions. FFN's are not able to transfer knowledge from one prediction to the next. In terms of predicting on sequential data where the previous prediction are related to the next this is a downfall. Recurrent Neural Networks(RNN) solves that issue by containing a "hidden state" which contains knowledge from previous time steps, this hidden state then feeds that information into the next prediction and is updated with new knowledge again.

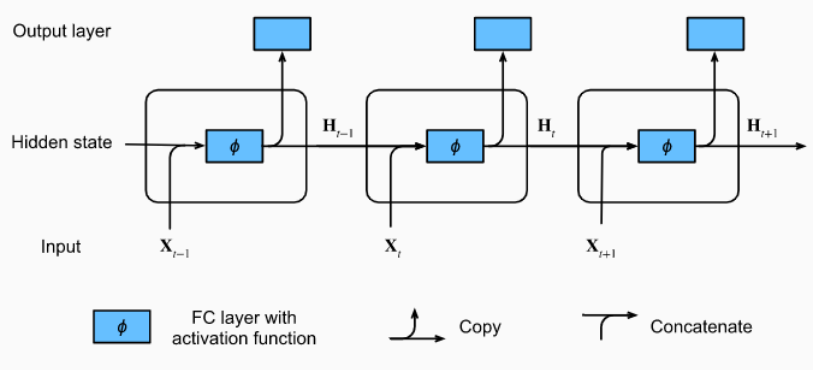


Fig. 13: Diagram of a RNN over three time steps.

Figure 13 shows the functionality of a RNN over a three time step cycle. Time step t the hidden state can be computed by: concatenating the input X_t and the hidden state H_{t-1}, from the previous time step, forwarding this information this result into a fully connected layer with the activation function ϕ . The output of such a layer is the hidden state H_t. At this moment the model parameters are the concatenation of the weights W_{xh} and W_{hh} and a bias of b_h. Including the parameters $W_{xh} \in \mathbb{R}^{d\times h}$, $W_{hh} \in \mathbb{R}^{h\times h}$ and biases $b_h \in \mathbb{R}^{1\times h}$, this can be expressed through a function:

$$H_t = \phi(X_t W_{xh} + H_{t-1} W_{hh} + b_h) \tag{4}$$

Thus the ouput of the model can be computed as:

$$O_t = H_t W_{hq} + b_q \tag{5}$$

where W_{hq} and b_q represents the weights and biases of the output layer. However an issue with the RNN is that it is not able to retain enough information to correctly capture the sequential relations for longer periods of time. The gradients of RNN's typically decays over time and this is called the vanishing gradient problem that regular RNN's suffer from. Therefore an extended RNN is needed, to fully capture the sequential information of the data.

6.4 Long Short Term Memory

Long Term Short Memory(LSTM) models aim to solve the issue of the vanishing gradient of RNN's by introducing a memory cell. The purpose of this memory cell is to extend the memory of the model even further than the RNN. The information that goes through the memory cell is controlled by different gates. Conventionally they are called the output gate, O_t, input gate I_t and forget gate F_t. the out- and input gates simply controls what comes in and out of the cell. The forget gate gives the cell the ability to reset its content in cases where it is best to ignore inputs.

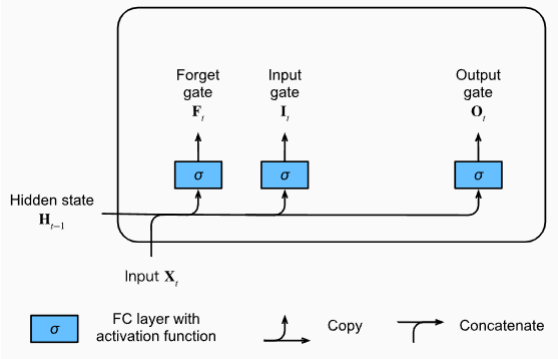


Fig. 14: Illustration of the computation done by the gates inside an LSTM

The data fed into the LSTM gates are the input at the current time step, X_t, and the hidden state from the previous time step H_{t-1}. This is processed through three layers with a sigmoid activation function, which calculates the values for each gate, 14. To express this mathematically assume: h hidden units, a batch size of n and the number of units is d. this gives the input $X_t \in \mathbb{R}^{n \times d}$, hiddenst state $H_{t-1} \in \mathbb{R}^{n \times h}$. The gates can be expressed as such:

What is 14?..

$I_t \in \mathbb{R}^{n \times h}$, $F_t \in \mathbb{R}^{n \times h}$, $O_t \in \mathbb{R}^{n \times h}$. Given this notation, the calculations looks like this:

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i) \tag{6}$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f) \tag{7}$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o) \tag{8}$$

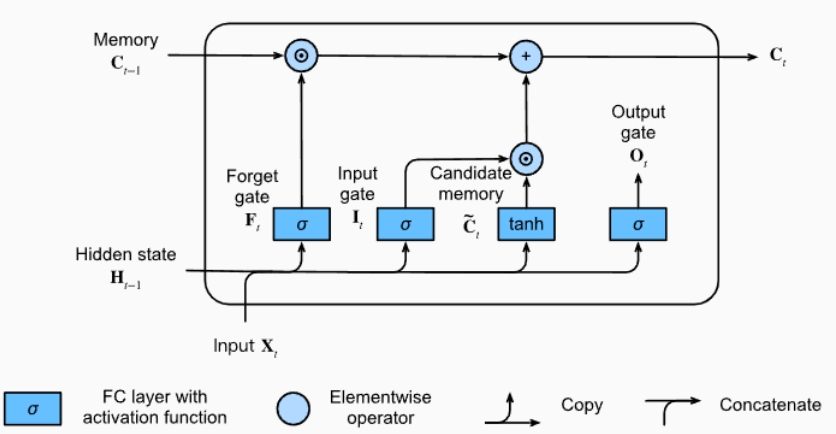


Fig. 9.2.3 Computing the memory cell in an LSTM model.

Fig. 15: Illustration of how the memory cell of the LSTM model is computed

The input and forget gates are controlled by two instances of a memory cell. The candidate memory cell $\tilde{C}_t$ controls the affect of new data from I_t. The forget gate controls how much information is kept from the previous memory cell, $C_{t-1} \in \mathbb{R}^{n \times h}$. Thus updating the memory cell leads to the following expression:

What is the meaning of this symbol?

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t \tag{9}$$

What does it mean that $F_t approximates 1?

If F_t approximates 1 and I_t approximates 0, C_{t-1} will be passed through to the current time step. This is how the LSTM model aims to overcome the vanishing gradient issue and better capture longer sequential dependencies. Finally the hidden state of the current time step, $H_t \in \mathbb{R}^{n \times h}$ needs to be computed. The current H_t is dependent on O_t and in the case of LSTM it is a gated tanh version of the memory cell, and is expressed as:

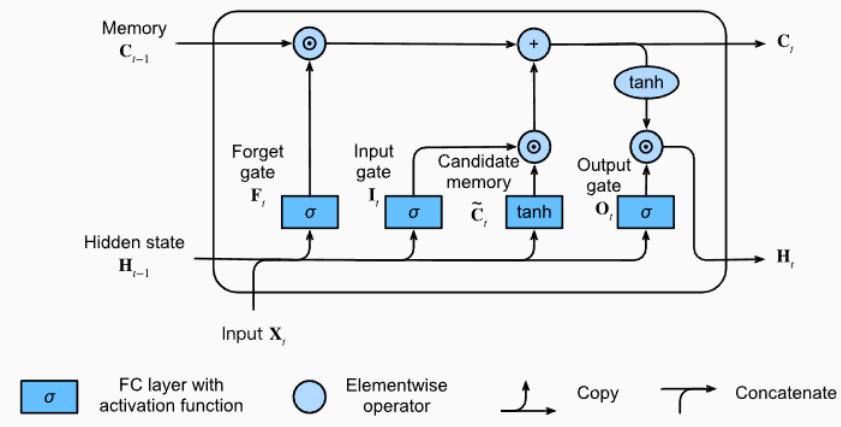


Fig. 16: Illustration of a complete LSTM model

$$H_t = O_t \odot tanh(C_t) \tag{10}$$

Chapter 6

Implementation

7 Error metrics

After having acquired an understanding of how the both XGBoost and LSTM's work, it is time to implement and train them on the data from Energinet and DMI. During training and parameter tuning of the models it is important to be able to have some form of error metric from which you can interpret the performance of the models. It is also important to choose an error metric that allows for comparison between models. For this project, two error metrics was chosen, Mean Squared Error(MSE), and the same as was used in the data section, R^2.
MSE was chosen since it gives an actual value for how much the models predictions deviate from the actual values. It is calculated with the following formula:

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

R^2 was chosen since it gives a value between 0 and 1, which is allows for easy comparison between different models. In simple terms, R^2 is interpreted as a percentage for how much of the variance in the dependent variable can be explained by the independent variables in the predictor. R^2 can be expressed as such:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y}_i)^2} \tag{12}$$

This section is about error metrics.

In order to understand the performance of the models during testing, the observed data from DMI will be split into two datasets. One for training and one for validation, this is generally best practice and an industry standard. In the case of this project the training data will be the data from 2010-2017 and the validation set will consist of the data from 2018 and 2019. This is the same year as the test set, which is the forecast data from the year 2019, but

I would suggest to give the exact dates.

Make it perfectly precise.

it should not be a problem when the data is not the same. This will actually give us a way of seeing how much the forecasts affect the performance of the models since the energy consumption is the same.
Other than these two measures for error and goodness of fit, the models will be compared to a ~~very~~ naive model or base line. Which will consist of taking the value for consumption from the same time, the day before. Expressed as the prediction made for time step $\hat{T}_{24(5-1)+8,20}$ will be compared to the actual value from $T_{24(4-1)+8,20}$. The naive model is to get a bare minimum threshold for the performance of the model, which should give a frame of reference for the implemented models performance.

7.1 Parameter Tuning of XGBoost

Achieving a well performing model in machine learning can sometimes be a tough task, dependent on many different aspects such as quality and amount of data, choosing the right model for the problem at hand. Afterwards, implementing the chosen model correctly can turn out to be quite the task as well. The goal is of parameter tuning is to make you model learn as much as possible on the amount of data you have available, but still avoiding that the model overfits on the training data and then performs very poorly on new data. Even though the XGBoost is perceived as being a model that is well performing "out of the box", it still contains parameters that needs to be tuned to arrive at the best performance. Choosing the parameters to tune was done by reading the documentation for the XGBoost, and then pick the ones that seemed relevant for this type of problem.
The first parameter was the max depth of the tree and the number of boosting iterations. These two are dependent on each other, since if you add more depth to the individual trees the model needs more iterations to generalize over the data. However, deeper trees have a higher chance of overfitting. next it was the step size shrinkage, or the learning rate, this is a parameter that aims to make the model more conservative and reduce overfitting. Then γ was chosen since it is the threshold of minimum loss required to make further partitions of the tree. Increasing γ makes the model more conservative. Lastly, the parameter minimum child weight was chosen, this sets the threshold for the minimum sum of weight needed in a child, this does also help regulate for overfitting. With 5 different parameters, that each will have different values that are optimal, this would require an extensive parameter search if it

Was some notation for these quantities introduced in the previous section?

was doing manually. However, since XGBoost is very fast to train and test, this was done in a nested for loop. A set of possible values were chosen for the model to go through for each parameter. The model then trained for all possible combinations and stored the results in a dataframe with the respective values for all parameters. This might sound like an extensive way of brute forcing the parameter search like this, but with the efficiency of the XGBoost it only took an hour to run it through. The benefit of running the parameters search like this, other than ensuring we get the best model, is that this allows for inspection of the impact of the parameters on the result.

Parameter settings used in parameter search				
Tree depth	Learning rate	γ	Min child depth	Iterations
6	0.01	4	3	100
8	0.03	6	5	250
10	0.1	8	8	500
				1000

Table 3: Data frame of the parameters that were set in the search for the best possible setup of the XGBoost model

Is this the same gamma?

Optimal parameter setups for XGBoost					
Zone	Tree depth	eta	Gamma	Min child depth	Iterations
DK1	10	0.01	4	8	500
DK2	10	0.03	6	8	100

Table 4: Optimal settings for the XGBoost model for the two zones DK1 and DK2

Was some notation for these quantities introduced in the previous section?

Looking at optimal parameter settings for both zones it would appear that increasing the tree depth further should increase the performance of the model. This was tested afterwards in separate tests and did not result in any increase in performance of the model. Thus the final parameter setup for the XGBoost model for DK1 and DK2 are presented in 4.

7.2 Encoder/Decoder LSTM model

As seen from the literature review, there are near endless directions one can go when setting up a neural network. With the most important feature of the training data being time, it was important to choose a setup that is able to account for that. Therefore, by suggestion from Energinet, the setup that was chosen for the model was an encoding and decoding LSTM setup. This allows

for the LSTM model to become bi-directional by taking in past values, and store the information from previous time steps in the memory cell and using it along with future time steps.

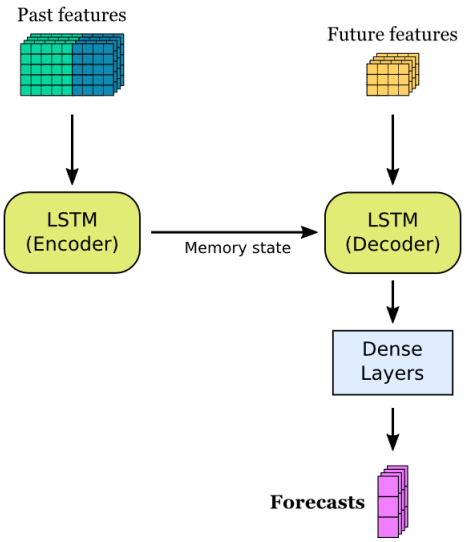
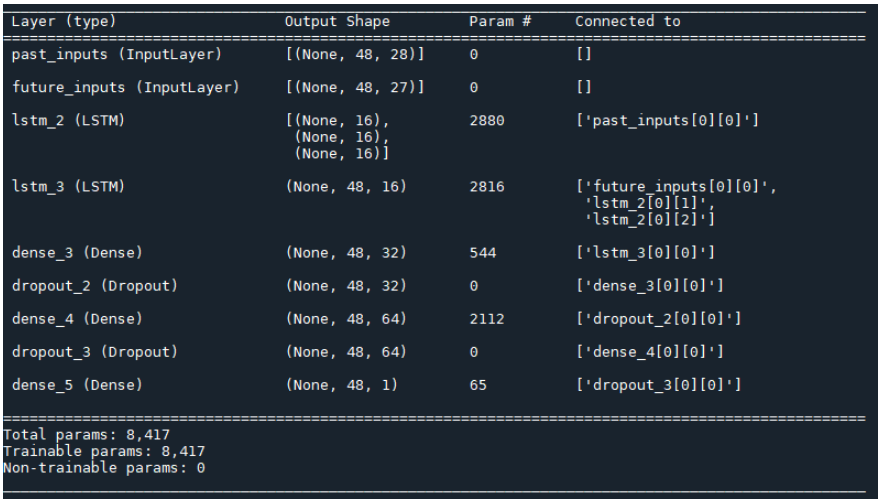


Fig. 17: Illustration of the encoder decoder implementation in the LSTM model

The implementation used in this setup is 48 hours backwards, and 48 hours forwards. This is a hyper parameter that will have to be set for the final LSTM model. Since it is difficult to set a preferable previous time step that is optimal for the model prior to training it. With a general structure of the LSTM, it is time to prepare the observed data for the purpose of being used in a NN. Two steps are needed; the data needs to be normalized between 0 and 1, and categorical values needs to be one-hot-encoded. It is widely considered best practice to normalize you data before using it with a NN, reasons being that it speeds up the training time and the way the activation functions inside the hidden layers calculate weights. The normalization of the dataset is done with package from SKLearn, called MinMaxScaler. This package just performs a regular minmax scaling on the dataset, but it lets us store the scaling parameters in a variable inside Python, which makes it easy to re-scale the predictions back to regular values again afterwards. The one-hot-encoding is performed on the feature for time of day. It is a categorical feature without ordinal relationship between the 24 values that represent the hours in a day. In practice this means instead of having one column for time, the dataset now contains 24 columns representing on hour each, containing binary values.



Layer (type)	Output Shape	Param #	Connected to
past_inputs (InputLayer)	[(None, 48, 28)]	0	[]
future_inputs (InputLayer)	[(None, 48, 27)]	0	[]
lstm_2 (LSTM)	[(None, 16), (None, 16), (None, 16)]	2880	['past_inputs[0][0]']
lstm_3 (LSTM)	(None, 48, 16)	2816	['future_inputs[0][0]', 'lstm_2[0][1]', 'lstm_2[0][2]']
dense_3 (Dense)	(None, 48, 32)	544	['lstm_3[0][0]']
dropout_2 (Dropout)	(None, 48, 32)	0	['dense_3[0][0]']
dense_4 (Dense)	(None, 48, 64)	2112	['dropout_2[0][0]']
dropout_3 (Dropout)	(None, 48, 64)	0	['dense_4[0][0]']
dense_5 (Dense)	(None, 48, 1)	65	['dropout_3[0][0]']

Total params: 8,417
Trainable params: 8,417
Non-trainable params: 0

Fig. 18: Table of the layers and parameters for the LSTM model

The two first layers of the model should be the encoding and decoding LSTM layers. These layers functions as the models input layer and has the windowed shape: $y_{encoder} \in \mathbb{R}^{48\times28}$ and $y_{decoder} \in \mathbb{R}^{48\times27}$, to fit the previous time steps and the future time steps without the consumption feature. In this implementation the output from the decoder layer is then fed into two consecutive dense layers containing the activation function called rectified Linear Unit, (ReLU). The output layer of the model is a 1-dimensional dense layer, also containing a ReLU activation function. In between all these layers are dropout layers with a 0.2 setting in order to combat overfitting. What dropout layers does is turning off a given percentage of the neurons during testing. This prevents the neurons from becoming dominant in specific cases and forces the weights of the other neurons to account for the information not captured by the turned off neurons.

Zone	Past input	Dropout	Activation function	Optimizer	Momentum	Learning rate	Loss function	Epochs
DK1	48	0.2	Relu	SGD	0.1	0.01	MSE	700
DK2	48	0.2	Relu	SGD	0.1	0.01	MSE	1000

Table 5: Parameter settings for LSTM Model

Next it was time to choose the optimizer for the model, here two different optimizers were tested. Firstly the ADAM was tested, but this would cause the

model to overfit instantly after one or two Epochs. Afterwards a Stochastic Gradient Descent(SGD) optimizer was implemented, which, at first also overfitted, but lowering the momentum of the SGD enabled to model to train for longer, with 0.1 ending up being the optimal setting for the model. The learning ended being a 0.01, smaller ones were investigated, but the training time would be increase significantly and would not converge.

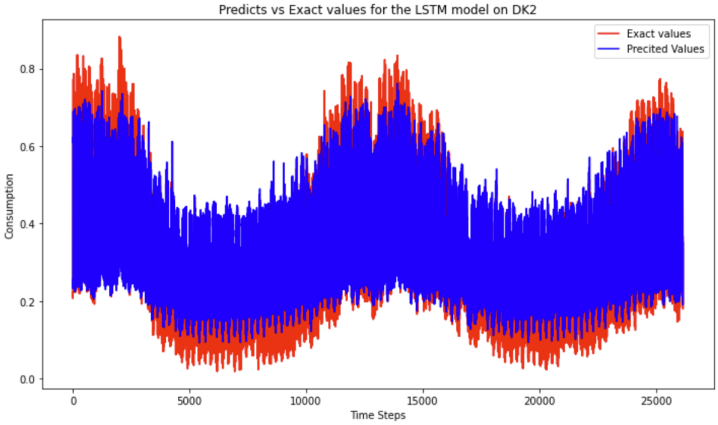


Fig. 19: Plot of the predictions done by the LSTM and the real values, for the validation set with early stopping enabled which did approximately 150 epochs

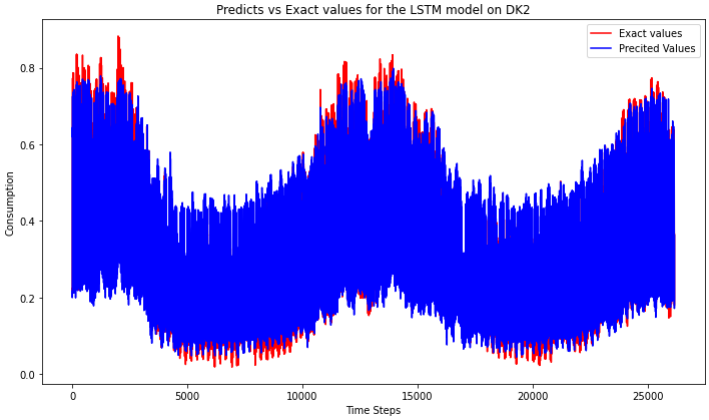


Fig. 20: Plot of the predictions done by the LSTM and the real values, for the validation set doing 1000 epochs without early stopping

Initially the model was also implemented with early stopping with a patience of 5. However, when the model would stop training at around 150-200 epochs it was not able to predict the maximum and minimum values for the validation set. From experience at Energinet, it was suggested that letting the model train as long as it does not start overfitting. Looking at the graphs in Figure

19 and 20, it is clear that letting the model for DK2 train for longer yielded a significantly increase to the performance of the model.

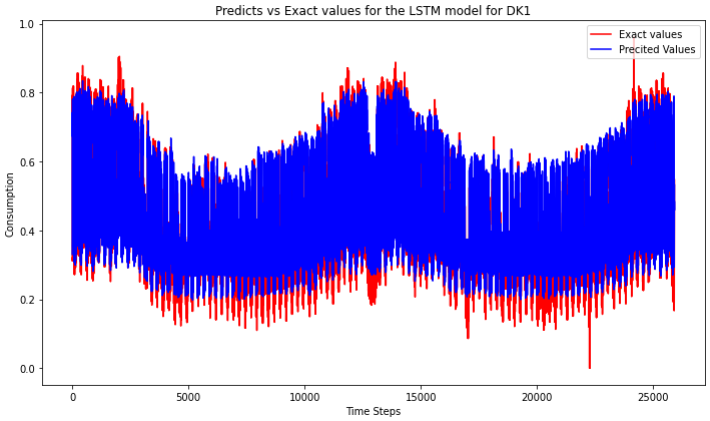


Fig. 21: Plot of the predictions done by the LSTM and the real values, for the validation set for the DK1 model.

For the DK1 implementation it started overfitting at around 700 epochs, and as it can be seen from Figure 21, the model was not able to achieve the same performance for DK1 as for DK2.

Validation Results Now that the parameters have been tuned, the error metrics can give insights to the performance of the models on the validation set.

Results of predictions on the validation set			
Model	zone	MSE	R^2
XGBoost	DK1	18645.2	90.5
LSTM	DK1	22104	88.7
Naive model	DK1	67384.9	64.7
XGBoost	DK2	7187.5	92.2
LSTM	DK2	7095.9	92.3
Naive model	DK2	13714.9	84

Table 6: Comparison of results for the XGBoost and the naive model

Inspecting the metrics from table 6, the performance of the XGBoost and LSTM models appears to be very close, for DK2 the difference is only 0.1 for the R^2 score. Looking at the R^2 for the naive model, for DK2, it seems to perform rather well. But from the MSE it is apparent that the trained models still almost halves the error value. Another interesting find from the table is the difference in the performance of the naive models. It would appear as if

the electricity consumption for DK1 has a much higher variance than the one for DK2. The variance for DK2 is 1554.8 and 2025.6 for DK1. So the difference in performance of the naive models does make sense, the positive take away from this is that both the LSTM and XGBoost is able to achieve similar performances despite the difference in the consumption pattern.

Chapter 7

Results

The predictions in this section of the project will be done solely on the forecast data for the year 2019. This is the data that the models will have to predict on in a real world scenario if it is implemented at Energinet. The error metrics for the models performance on this data will be same metrics used in previous predictions to ensure the new performance metrics will be relatable to the earlier ones.

Results of predictions on the test set			
Model	zone	MSE	R^2
XGBoost	DK1	33216.7	82.7
LSTM	DK1	22720.2	89.1
Naive	DK1	111646.9	41.8
XGBoost	DK2	9279.9	89.2
LSTM	DK2	8817.1	90.17
Naive	DK2	36152.9	57.8

Table 7: Performance metrics for the models on the test set

The results for the test set is very similar to the results for the validation set. Except for the XGBoost model for DK1, which dropped almost 8% in R^2 score, even though the LSTM model stayed at a similar performance as with the validation set. This indicates that the LSTM model is more robust and better generalized than the XGBoost. Because the poor quality that was seen from the graph for the radiation feature from the forecast data, there was some concern of the affect that this might have on the performance of the models. This could be the reason for the drop in performance of the XGBoost model for DK1.

Chapter 8

Discussion and Further Work

With the datasets that are getting released from DMI now, it would make sense to revisit the training data when the Climate data set is released. Having a completely cleaned dataset versus a dataset containing watered down outliers, it will be better to have the cleaned dataset, but the lack of data in that dataset and missing values in that dataset will need to be fixed before it is used.

What is this about?

Looking at the precision of the forecasts coming from DMI on radiation, and the SHAP values from the XGBoost model, it would make sense to do the same predictions without this feature. As stated in the section for the forecast data, your model can only get as good as your data allows it to be. If the same predictions were done without the radiation it would give insight if this feature is contributing to the performance of the models at all. However, since the feature is not very important at all for the model, the low precision of the forecasts might not be so detrimental to the models as one could think. Nevertheless, this would be an interesting topic for further work on the project.

It should be simple to remove one predictor and train the model without it.

Looking at how precise the temperature feature is from the forecast data, and how the radiation feature tends to underestimate the radiation, it would be interesting to see the performance of the models if trained on the forecast data instead of observed values. If the plot of the forecast data is indicative of previous year's predictions as well, then it would be interesting to see the models performance if trained on this data instead. However, this would only make sense if the above mentioned idea, of excluding the radiation is tried beforehand. reason being, if the radiation feature does not benefit significantly to the model, it should just be dropped and there would be no reason of training the models on the forecast data anyway.

Why?

The performance of the XGBoost came as a surprise, however it shows that the default solution for regressional forecasts. It is worth taking into consideration that the XGBoost model takes under a minute to train, where as the LSTM takes 8 hours. Is that 1% worth the computational cost, or would it suffice to settle for the XGBoost? Also the implementation process of the XG-

I think one-hot encoding is also needed for XGBoost (correct me if I am wrong).

Boost is significantly less time consuming than the LSTM. For the LSTM, the data has to be structured so it can be windowed, then normalized and one-hot encoded, and lastly the predictions will have to be fetched from the windowed data structures and sorted. The XGBoost just takes the input as it is and predicts on that single time step without requiring any further preprocessing.

Chapter 9

Conclusion

References

Antón Román-Portabales Martín López-Nores, José Juan Pazos-Arias (2021). *Systematic Review of Electricity Demand Forecast Using ANN-Based Machine Learning Algorithms*. DOI: https://doi.org/10.3390/s21134544.

Energihåndbogen (2019). URL: https://evu.dk/wp-content/uploads/2019/06/Graddage.pdf.

Energistyrelsen (2019). *Energistatistik*, pp. 34–39. URL: https://ens.dk/sites/ens.dk/files/Statistik/energistatistik2019_dk-webtilg.pdf.

Friedman, Jerome H. (1999). "GREEDY FUNCTION APPROXIMATION: A GRADIENT BOOSTING MACHINE". In: pp. 1189–1232.

Hristos Tyralis, Georgia Papacharalampous (2021). *Boosting algorithms in energy research: a systematic review*. DOI: https://doi.org/10.1007/s00521-021-05995-8.

Iman Ghalehkhondabi Ehsan Ardjmand, Gary R Weckman (2016). *An overview of energy demand forecasting methods published in 2005–2015*. DOI: https://doi.org/10.1007/s12667-016-0203-y.

Jason Runge, Radu Zmeureanu (2021). *A Review of Deep Learning Techniques for Forecasting Energy use in Buildings*. DOI: https://doi.org/10.3390/en14030608.

Scott M. Lundberg, Su-In Lee (2017). *A Unified Approach to Interpreting Model Predictions*. URL: https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

Tianqi Chen, Carlos Guestrin (2016). *XGBoost: A Scalable Tree Boosting System*. DOI: https://doi.org/10.48550/arXiv.1603.02754.

Ziel, Florian (2018). *Modeling public holidays in load forecasting: a German case study*. DOI: https://doi.org/10.1007/s40565-018-0385-5.