Pål Christian Glenna Iversen
Håkon Thorstensen

Automatic Wind Power Forecasting as a Service

Master's thesis in Computer Science
Supervisor: Odd Erik Gundersen
June 2020

Abstract

Time series forecasting is a highly relevant and valuable tool for a wide range of businesses applications and domains. However, due to the vast technical challenges and required expertise to optimize modern forecasting techniques, many businesses lack the in-house competence required to achieve good results. In this project, we investigate and experimentally develop an automatically scaling, machine learning based forecasting service for wind power forecasting. The system uses modelling techniques tailored for wind power forecasting, in order to compete with more general AutoML services, which provides a similar value to end users.

A model-based transfer learning strategy of pre-training, testing, pruning and refining is used with gradient tree boosting to leverage data from other wind farms. The use of transfer learning improve forecasting accuracy and provide valuable predictions even for wind farms with little available data. We also test several strategies for utilizing the multitude of source wind farms available, and we test two strategies for weighting source wind farms based on similarity and transfer performance.

The system itself is implemented with a focus on scalability and modifiability, with the specific purpose of planning for future challenges. To achieve these qualities our primary focus is on the overall system architecture, and in particular how a software system could be designed to support machine learning processes. We utilize concepts such as microservices, cloud computing, automation and technical debt reduction.

Through practical and analysis based experiments we demonstrate that it is possible to create a system that automates the use of wind power forecasting, even for wind farms with little available data. We show that the use of transfer learning with gradient tree boosting gives consistent improvements in forecasting performance across a wide range of forecasting horizons and target training data availability. Particularly for small available data quantities, transfer learning provides significant improvements. The tested weight generation strategies fails to provide substantial improvements to model performance compared to equal weights. We also perform experiments to demonstrate that the system can scale to support training of multiple models and accommodate increased user load. These experiments show that the system can accommodate at least 500 concurrent users, and that the training time is not vastly affected by whether the system trains 10 or 20 project. Finally, based on an analysis of our modifiability requirements, we also demonstrate that the system should be easy to modify to accommodate future requirements.

Preface

This report represent the the findings for our master's thesis conducted at the Norwegian university of science and technology (NTNU), under the department computer science. The work was performed by two fifth year computer science students; Pål Christian Glenna Iversen, who specializes in software development, and Håkon Thorstensen, who specializes in artificial intelligence. The work was conducted in collaboration with the AI department at TrønderEnergi, that provided domain knowledge and test data to conduct our experiments.

We would like to thank our supervisor Odd Erik Gundersen who is an Associate Professor at the NTNU and chief AI officer at TrønderEnergi, for his interest and guidance. We would also like to thank Gleb Sizov who is an AI engineer at TrønderEnergi, for his feedback and help during the project.

Pål Christian Glenna Iversen

Håkon Thorstensen

Trondheim, June 10, 2020

Contents

List of Figures

List of Tables

Abbreviations

Symbol	=	**Definition**
ACR	=	Azure Container Registry
AKS	=	Azure Kubernetes Service
ANN	=	Artificial Neural Network
API	=	Application Programming Interface
ARIMA	=	AutoRegressive Integrated Moving Averages
ASGI	=	Synchronous Server Gateway Interface
ATL-DNN	=	Adaptive Transfer Learning in Deep Neural Networks
AutoML	=	Automated Machine Learning
CaaS	=	Containers as a Service
CACE	=	Changing Anything Changes Everything
CD	=	Continuous Delivery
CD/CDE	=	Continuous Deployment
CI	=	Continuous Integration
CSRF	=	Cross Site Request Forgery
CSS	=	Cascading Style Sheets
DA	=	Domain Adaption
DBN	=	Deep Belief Network
DNN	=	Deep Neural Network
DNN-MRT	=	Deep Neural Network based Meta Regression and Transfer learning
DSL	=	Domain Specific Language
ES	=	Exponential Smoothing
ES-RNN	=	Exponential Smoothing Recurrent Neural Network
FaaS	=	Forecasting as a Service
FFORMA	=	Feature-based FORecast Model Averaging
GRU	=	Gated Recurrent Unit
GTB	=	Gradient Tree Boosting
GUI	=	Graphical User Interface
HTML	=	HyperText Markup Language
HTTPS	=	HyperText Transfer Protocol Secure
IaaS	=	Infrastructure as a Service
IBT-GBDT	=	Instance Based Transfer - Gradient Boosted Decision Trees
ISO	=	Organization for Standardization

Symbol	=	**Definition**
JSON	=	JavaScript Object Notation
LSTM	=	Long Short-Term Memory
MAE	=	Mean Absolute Error
MLP	=	Multi-Layer Perceptron
MSDA	=	Multi-Source Domain Adaption
MTL	=	Multi-Task Learning
MVC	=	Model View Controller
ORM	=	Object-relational mapper
PaaS	=	Platform as a Service
RBM	=	Restricted Boltzmann Machine
ReLU	=	Rectified Linear Unit
REST	=	Representational State Transfer
RMSE	=	Root Mean Square Error
RNN	=	Recurrent Neural Network
SaaS	=	Software as a Service
SDK	=	Software Development Kit
SQL	=	Structured Query Language
TCN	=	Temporal Convolutional Network
TL	=	Transfer Learning
TL-GTB	=	Transfer Learning - Gradient Tree Boosting
TLS	=	Transport Layer Security
URL	=	Uniform Resource Locator
WSGI	=	Web Server Gateway Interface
XSS	=	Cross Site Scripting

Chapter 1

Introduction

Within the energy sector, a wide range of uncertain energy producers and consumers exists. Predicting the amount of energy produced or consumed by these entities, allows for better resource management and market trading. While simple methods of predicting production and load in energy systems are widely utilized, Foley et al. [18] demonstrate that more accurate predictions can be achieved by using machine learning. However, many power companies lack the in-house competence to create such models with good results.

This creates the demand for a software as a service approach(SaaS), where the power companies delegate the modelling process to a competent third party. Within the energy sector, such a service can be provided for specific problems, for example wind power production. The service needs to handle a large number of problem instances that need to be solved, with related data available. Several challenges for such a service needs to be addressed, in terms of scaling infrastructure with problem instances, minimizing modelling overhead for each problem instance and maximizing model accuracy across all problem instances.

Several leading cloud providers offer automated machine learning(AutoML) as a service. According to Feurer et al. [17] AutoML can give good results with limited machine learning expertise. Existing AutoML can be applied to any problem, however the result is usually not as good as a modelling solution manually tailored to the forecasting problem. We believe there is a relevant middle ground between AutoML and manual modelling, where an automatic modelling service can be created and optimized for a specific problem type, providing both the automatic scalability of AutoML and the tailored performance of manual modelling.

Such a system will have access to many different instances of similar problems, which motivates the use of transfer learning to improve model performance. Transfer learning is a strategy within machine learning, where data and/or learned knowledge from similar source problems are leveraged to improve model performance on a target problem. This is analogous to knowledge transfer in human learning, which is a critical aspect of the generalizability and flexibility of human intelligence. It is for example easier for a human learner to learn Japanese if they can already speak Chinese, compared to if they can only speak western languages. Transfer learning can particularly facilitate good model performance with little to no available training data on the target problem.

During this project, we investigate a fully automatic system to handle the creation and modelling of new problem instances for time series forecasting for wind power production. Through experimental development we utilize state of the art techniques and processes, to develop a highly competitive system. The system allows users to upload data for a wind farm through a web application and have a model optimized for their wind farm automatically trained for them. This training procedure uses transfer learning to leverage data from other wind farms to improve performance. The user is able to use this model for future forecasts for their own application needs, by accessing it through a REST API.

In physical energy markets, expected production must be reported the day before, for a 24 hour window. This expected production is reported 12 hours before the start of this window. Additionally, there might be a delay in the availability of power production data. The system therefore provides wind power forecasts for 48 hours forwards in time.

The project is carried out in two parts spanning one term each. The first term was to conduct a literature review and the creation of simple experiments to ensure the feasibility of such as system. This report covers the results of the second term and will consist of a complete system implementation and a thorough investigation into the effectiveness of transfer learning on wind power forecasting. The project is carried out in cooperation with TrønderEnergi, a green energy company situated in Trondheim, Norway. Our system is designed with the existing knowledge and practices of TrønderEnergi in mind, leading us to focus on a python centered implementation built on Microsoft's Azure cloud infrastructure. TrønderEnergi provides datasets for experimentation on wind power forecasting and this report will serve as technical documentation for TrønderEnergi, to help maintain the system.

Based on the purpose of the project, we propose the following hypothesis and research questions:

HYP: *An automatic, scalable machine learning service can be developed for wind power forecasting for wind farms with little available data.*

RQ1: *How can transfer learning be applied to improve wind power production forecasting performance?*

RQ2: *How can a system be designed to automate the use of transfer learning for wind power forecasting?*

RQ3: *How can a system for wind power production forecasting utilizing transfer learning be designed to support modifiability and scalability?*

The hypothesis and research questions are designed with the the purpose of identifying whether it is possible to create a system that can utilize transfer learning and system engineering practices inr order to compete with existing solutions. RQ1 is concerned with the adaption of transfer learning to the specific problem domain of wind power forecasting, and whether or not it can be used to improve the performance of such a system. This is of particular interest for wind farms with little to no target data. RQ2 is concerned with the system implementation of transfer learning, and how a transfer learning solution can be integrated into a lager system. Finally RQ3 is concerned with the quality attributes of the system, namely modifiability and scalability. These quality attributes are selected to support the future development and ensure that the system can be adapted to future demand. Modifiability was selected to ensure that the system can be changed according to the needs of TrønderEnergi. This could for instance be to add new machine learning methods or even support forecasting of other time series related problems. Scalability was added to ensure that the system can scale to accommodate a larger user base, particularly in regard to simultaneously training multiple wind farms.

This report is structured as follows: Relevant background theory for the project is presented in chapter 2. Related works and the most relevant findings of the literature review are presented in chapter 3. The transfer learning methods to be tested are presented in chapter 4. The system design, implementation and architecture is described in chapter 5. The experiments and results are presented in chapter 6 and chapter 7. Finally, an evaluation of our work and suggestions for future work are covered in chapter 8.

Chapter 2

Background Theory

To design a system that has good forecasting performance, while also being modifiable and scalable; interdisciplinary knowledge is essential. This chapter will introduce the basis for time series based machine learning problems and software engineering techniques and practices. The topics to be explained will provide fundamental knowledge of the problem domain, and is meant to give context for our research.

2.1 Time Series Forecasting

Time series forecasting [24] is the prediction of the future value of a time-varying variable at one or more times in the future. Time series forecasting encompasses anything from weather forecasting to market prognosis, and have been the subject of a great deal of interest and research.

A time series forecast can either be univariate, where only the value future value of a single variable is predicted, or multivariate, where multiple target variables are predicted. The forecasting might use only the variables that are forecasted, or other, related time-dependent, exogenous variables. A distinction is also made between single-step forecasting, in which the value for only the next timestep is forecasted, and multi-step forecasting, where forecasts are made for a forecasting horizon of multiple timesteps forwards. Wind power forecasting is univariate, as power production is predicted, uses weather forecasts as exogenous variables, and we are interested in forecasts for multiple hours ahead, making it multi-variate.

Forecasting entails predicting the value of target variable $y_{i,t}$ for entity i at time

t. Different entities correspond to different groupings of time series data, and in our case corresponds to different wind farms. Such a forecast can be expressed as:

$$\hat{y}_{i,t} = f(\mathbf{y}_{i,t-h-k:t-h}, \mathbf{x}_{i,t-h-k:t-h}, \mathbf{s_i})$$

Where $\hat{y}_{i,t}$ is forecast, $\mathbf{y}_{i,t-h-k:t-h} = \{y_{i,t-h-k}, ..., y_{i,t-h}\}$ are past observations for the value to be predicted, $\mathbf{x}_{i,t-h-k:t-h} = \{\mathbf{x}_{i,t-h-k}, ..., \mathbf{x}_{i,t-h}\}$ are past observations for the exogenous variables, h is the forecasting horizon, k is the size of the input window of each variable and s_i is a set of static metadata for the specific entity. $f(\cdot)$ is the predictive function, implemented by a forecasting model.

Modern approaches for time series forecasting generally falls into one of the three categories: statistical methods, machine learning methods or hybrid approaches.

2.1.1 Statistical Approaches

Classical, statistical approaches to time series forecasting utilize simple statistics across past data to predict future values. A couple of notable examples: *Exponential smoothing* (ES), which uses a window of past values with exponentially decaying weights to extract level, trend and seasonality used for predicting future values. *Autoregressive integrated moving averages* (ARIMA), which differenciates the data for stationarity, uses regression on past values and past residuals then reintegrates the result.

2.1.2 Machine Learning Approaches

Machine learning approaches treats time series forecasting as a supervised regression learning problem, aiming to learn a mapping from known observations to future time series values. This mapping is learned automatically by a machine learning algorithm, utilizing historical data for the time series to be forecasted, and possibly data from related problems through transfer learning.

2.1.3 Hybrid Approaches

Hybrid approaches to time series forecasting aim to combine aspects of classical, statistical time series forecasting methods with machine learning. A common approach is to use statistical methods to preprocess the time series, like detrending and deseasonalizing, then predicting only the level component of the time series using machine learning. Another alternative is integrating aspects of statistical forecasting methods directly into a machine learning model, trained as an end-to-end system.

2.2 Machine Learning Methods

Supervised machine learning regression utilizes general machine learning algorithms to automatically learn underlying statistical relationships from data in order to create a predictive model. Given a feature space $\mathcal{X}$ and a label space $\mathcal{Y}$, a predictive function $f : \mathcal{X} \rightarrow \mathcal{Y}$ is learned using a set of N training examples $\{(x_1, y_1), ..., (x_N, y_N)\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. When applied to time series forecasting, $\mathcal{X}$ consists of previous time series values and related data, such as weather forecasts, while $\mathcal{Y}$ consists of future time series values. The predictive function $f(\cdot)$ is learned to minimize the prediction error on future predictions, using a machine learning algorithm.

2.2.1 Artificial Neural Networks

Artificial Neural Networks are computational models inspired by the distributed nature of the brain [46, pp. 727-737]. The core idea behind neural networks, is the use of many, simple computational units connected to each other in a larger network, which can represent complicated computational functions. Many specialized computational units exists, and with a wide range of possible network structures and the ability to flexibly combine different types of units into one network. Consequently, neural networks encompasses a wide range of different modeling approaches well suited for many different learning problems.

Network Structure

A neural network is typically structured into layers, where each layer consists of one or more units. The units in each layer take the output of the previous layer as input, and the output of the layer is a vector of the output of all the units within the layer. Each layer implements a transformation that is applied to the input vector, giving a transformed output vector, starting with x as the input to the network and ending in the prediction $\hat{y}$ as output. In between each layer, the data goes through different *internal representations*, which gradually extract information relevant for the final prediction.

Activation Functions

Activation functions are simple functions that map an input value to an output value. Activation functions are used within many different units, with the primary purpose of introducing non-linearity into the network. Without such non-linearities, parts of or the entire network might consist of only linear operations, and can therefore only represent linear functions. Many different activation functions exists, the most commonly used being the *Rectified Linear Unit* (ReLU), the sigmoid function (σ) and the hyperbolic tangent function (tanh).

$$ReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Standard Neuron

The simplest and most commonly used units in neural networks are standard neurons. Fully connected layers of such units are often referred to as dense layers and a neural network consistent of only dense layers are also called multi-layer perceptrons (MLP). Standard neurons take a weighted sum of its inputs plus an internal bias, transforms it through an activation function and passes the result as its output. The weights and bias are learnable parameters, which are optimized for the learning problem during training. Mathematically, standard neurons can be expressed as follows:

$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

Where y is the output of the neuron, n is the number of inputs, corresponding to the number of units in the previous layer for fully connected layers, w_i is the learnable weight for the i-th input, x_i is the i-th input, b is the learnable bias and f is the activation function.

Long Short-Term Memory

Long Short-Term Memory (LSTM), originally proposed by Hochreiter and Schmidhuber [20], is a type of recurrent neural network unit that adds an internal memory state to the network. Recurrent units are units in which the units output from the previous run of the network is also fed as input. This recurrent loop creates an internal memory state that allows the network to learn to model dependencies across examples. Recurrent neural networks is a natural fit for problems where there is a dependency between subsequent examples, as is the case in time series data. Recurrent loops can be implemented with standard neuron units, however these networks struggle to learn and remember long term dependencies. LSTM units were developed to better capture both the long and the short term dependencies in data.

In addition to taking the output of all LSTM units within the layer as input at the timestep, an LSTM unit has a second, cell state which is also passed across timesteps. The cell state is manipulated through three gates, the *input gate*, the *output gate* and the *forget gate*. The input gate regulates how information from a new input is added to the cell state. The forget gate regulates whether information is kept in the cell state and the output gate regulates how cell state information is used for computing the output. Mathematically, an LSTM unit can be expressed as follows:

$$\begin{aligned}
f_t &= \sigma(\sum_{i=1}^{n} w_i^f x_{t,i} + \sum_{j=1}^{m} u_j^f h_{t-1,j}) \\
i_t &= \sigma(\sum_{i=1}^{n} w_i^i x_{t,i} + \sum_{j=1}^{m} u_j^i h_{t-1,j}) \\
o_t &= \sigma(\sum_{i=1}^{n} w_i^o x_{t,i} + \sum_{j=1}^{m} u_j^o h_{t-1,j}) \\
c_t &= f_t c_{t-1} + i_t \tanh(\sum_{i=1}^{n} w_i^c x_{t,i} + \sum_{j=1}^{m} u_j^c h_{t-1,j}) \\
h_t &= o_t \tanh(c_t) \\
c_0 &= 0, h_0 = 0
\end{aligned}$$

Subscript t denotes the timestep. f_t is the forget gate activation, i_t is the input gate activation and o_t is the output gate activation. n is the number of inputs to the unit and m is the number of units within the LSTM layer. w_i^f, w_i^i, w_i^o and w_i^c are learnable weights for the i-th input and $x_{t,i}$ is the current i-th input. u_j^f, u_j^i, u_j^o and u_j^c are learnable weights for the previous output of j-th LSTM unit and $h_{j,t-1}$ is the previous output of the j-th LSTM unit. σ is the sigmoid activation function and tanh is the tanh activation function.

Training

Neural networks typically learn parameter values from data by utilizing error gradients. For a set of training data the network is used to predict target values for each example. The resulting prediction is compared to the true labels of the data using a loss function, most commonly *root mean square error* (RMSE) or *mean absolute error* (MAE) for regression problems. The gradient of this loss function of all trainable parameters within the network is computed and used to update parameter values in the opposite direction of the gradient, in order to reduce the loss. The simplest, gradient descent update rule is as follows:

$$\boldsymbol{p_{i+1}} = \boldsymbol{p_i} - \eta \nabla L(\boldsymbol{p_i})$$

Where $\boldsymbol{p_i}$ is a vector of all trainable parameters within the network. $L(\boldsymbol{p_i})$ is the loss, in this case as a function of trainable parameters, given a fixed set of training data. η is the learning rate, which is a tunable hyperparameter of the model. $\boldsymbol{p_0}$ is typically initialized to a random vector.

Due to the risk of this optimization method getting stuck in suboptimal local minima, more complex gradient based optimization algorithms have been developed, such as ADAM and RMSprop. These methods use concept such as momentum and decay to better balance the exploration-exploitation trade-off during training.

A problem known as the vanishing gradient problem arises in deep neural networks with many layers. As the error gradient is calculated from the output layer and propagated backwards through the network, the gradient tends to become smaller and smaller. For deep networks, this results in early layers receiving a gradient that is too small for learning optimal weights from random initialization. A common approach to resolving the vanishing gradient problem, is by utilizing layer-wise unsupervised pre-training of the network. A common method is to pre-train each hidden layer as the hidden layer in a three-layer autoencoder. The autoencoder is trained to recreate its own input, and is trained on the output of the previous layer in the deep network. Alternatively, a similar pre-training strategy uses a stack of *Restricted Boltzmann Machines* (RBM), which are two-layer networks trained to recreate model input through a backward pass through the network. Deep neural networks pre-trained with stacked RBMs are known as *Deep Belief Networks* (DBN). These pre-trained networks are then fine-tuned on labeled data using backpropagated gradients to create the final, predictive model.

2.2.2 Decision Trees

Decision trees are a simple and intuitive, yet powerful modelling approach used in machine learning [46, pp. 697-707]. Decision trees are trees where each internal node represents conditional decisions on input data in a flowchart-like manner. For each example, the tree is traversed from the root and down, with each decision indicating what child node to traverse to. The final, leaf node indicates the final prediction for the example. Decision trees can be used for regression problems with both numerical features and numerical labels. In such a case the conditional decision made in each internal node will be in the format $x^j > s$ where x^j is the j'th feature of example x and s is a numerical split point numerical threshold.

Decision trees can represent a wide range of predictive functions while excelling at intuitiveness and explainability.

Decision Tree Learning

Several different algorithms for decision tree learning exists, but most build on the idea of recursively building the tree by splitting the training data to best separate label values. Depending on whether features are categorical or numerical, different strategies need to be employed for choosing the splitting feature and condition. Depending on whether labels are categorical (classification trees) or numerical (regression trees), different strategies for leaf node values and split quality metrics needs to be used. To illustrate the learning of a decision tree, the *Classification and Regression Tree* (CART) algorithm for learning a binary regression tree is described.

In CART, a decision tree is learned in a greedy, top-down recursive manner. Starting with a dataset $D = \{(x_i, y_i)\}_{i=1..n}$ with n examples. Given a splitting feature j and split point s, the dataset is split into two sets as follows:

$$R_1(j,s) = \{x \in D \mid x^j \leq s\}$$
$$R_2(j,s) = \{x \in D \mid x^j > s\}$$

The goal of one split is to find the optimal split feature k and split point s that minimizes the loss metric, which is sum of squares for CART. That is, solving:

$$\arg\min_{j,s} \left(\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right)$$

where c_1 is the predicted value for the examples in $R_1(j,s)$ and c_2 is the predicted value for the examples in $R_2(j,s)$.

The inner minimizations are easily solved as the average of the label for the data in each respective partition of the dataset:

$$c_1 = \frac{1}{|R_1(j,s)|} \sum_{x_i \in R_1(j,s)} y_i$$
$$c_2 = \frac{1}{|R_2(j,s)|} \sum_{x_i \in R_2(j,s)} y_i$$

The outer minimization is solved by testing all splitting features j and all unique values for x^j in D as splitting point s.

After finding optimal values for j and s, an inner decision node in the tree is generated based of these values, for each child is either made into a leaf node the corresponding optimal prediction $c1$ or $c2$ if a stopping criteria has been met, for example max depth or 0 error. Alternatively a new split is generated as the child node by running this procedure recursively on the partitioned dataset given by $R_1(j,s)$ or $R_2(j,s)$

Gradient Tree Boosting

Gradient Tree Boosting is an ensemble method that combines multiple decision trees to create more complex and more accurate models. Gradient boosting can be applied to any base model, not only decision trees, but decision trees is the most commonly used base model in practice.

Gradient Tree Boosting builds an ensemble of decision tree models in a gradient descent fashion, by training new models on the negative gradient of the loss function in respect to the predictions of the previous set of models in the ensemble.

Given a set of training data $D = \{(x_k, y_k)\}_{k=1..n}$ with n examples, a gradient boosting ensemble predictor $F_M(\cdot)$ consistent of M base models is trained iteratively. At iteration $1 \leq m \leq M$ a training set for a new base model is generated based on the partially built ensemble $F_{m-1}(\cdot)$:

$$D_m = \left\{ (x_k, -\frac{\partial L(y_k, F_{m-1}(x_k))}{\partial F_{m-1}(x_k)}) \right\}_{k=1..n}$$

Where $L(y,x)$ is a loss function, for example MAE or RMSE.

A new base predictor $h_m(\cdot)$ is trained learned from the dataset D_m, and added to the ensemble:

$$F_m(x) = F_{m-1}(x) + \eta h_m(x)$$

Where η is the learning rate.

This method of utilizing gradient descent for constructing an ensemble is similar to the usage of gradient descent described in subsection 2.2.1, where neural network parameters are gradually shifted and optimized to minimize the loss function. In the case of gradient boosting, it is the predictions that are shifted to minimize the loss function, using base models trained on the negative gradient to ensure generalization to new examples.

The gradient tree boosting implementation CatBoost presented by Dorogush et al. [15], has been the most successful modelling method in TrønderEnergi's previous experiments with wind power forecasting.

2.2.3 K-means Clustering

Clustering is a different problem type than forecasting. The goal of clustering is to identify groups, or so called clusters, of similar examples within a set of data. New examples can then be assigned to one of these clusters. *k-means clustering* is a common machine learning approach for clustering.

Given observations $x_1, ..., x_n \in \mathbb{R}^d$ and initial, randomly chosen clusters centers $c_1, ..., c_k \in \mathbb{R}^d$, a two-step iterative process is used to re-compute the cluster centers.

In the **assignment step**, each observation is assigned to the closest cluster center according to some distance metric, usually squared eucledian distance:

$$a_i := \arg\min_j \|x_i - c_j\|^2$$

Then in the **update step**, the cluster centers are updated to be the average position of all observations assigned to it:

$$c_j := \frac{\sum_{i=1}^n [a_i = j] x_i}{\sum_{i=1}^n [a_i = j]}$$

Which is expressed using Iverson bracket notation, where $[P]$ is 1 if proposition P is *true* and 0 otherwise.

Repeated steps of assignment and cluster center updating is repeated until convergence, which occurs when no cluster center is moved during an update step.

2.3 Transfer Learning

In traditional machine learning approaches, a critical assumption is that all the training data is sampled from the same underlying distribution as the data the model will be applied to. Transfer learning seeks to surpass these limitations, and leverage training data from similar, but nonidentical distributions to learning the target distribution.

2.3.1 Formal Definition and Setting

Pan and Yang [41], introduces a formal definition of transfer learning, based on the concept of *domain* and *task*. A domain $D = \{\mathcal{X}, P(X)\}$ consists of a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. A task $T = \{\mathcal{Y}, f(\cdot)\}$ consists of a label space $\mathcal{Y}$ and a predictive function $f(\cdot)$, which is learned from training data consisting of pairs (x_i, y_i) where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$.

Pan and Yang defines transfer learning as follows:

> Given a source domain D_s and learning task T_s, a target domain D_t and learning task T_t, *transfer learning* aims to improve the learning of the target predictive function $f_t(\cdot)$ in D_t using knowledge in D_s and T_s, where $D_s \neq D_t$ or $T_s \neq T_t$.

Further, Pan and Yang [41] specifies different settings transfer learning based on the relationship between the source and target domain and task (Table 2.1) and data availability (Table 2.2).

Learning Setting		Source and Target Domains	Source and Target Tasks
Traditional Machine Learning		The Same	The Same
Transfer Learning	Inductive Transfer Learning / Unsupervised Transfer Learning	The Same	Different but Related
		Different but Related	Different but Related
	Transductive Transfer Learning	Different but Related	The Same

Table 2.1: Settings for traditional machine learning and transfer learning given domain and task similarity (adapted from [41])

Transfer Learning Setting	Related Areas	Source Domain Labels	Target Domain Labels	Tasks
Inductive Transfer Learning	Multi-task Learning	Available	Available	Regression Classification
	Self-taught Learning	Unavailable	Available	Regression Classification
Transductive Transfer Learning	Domain Adaption Sample Selection Bias Co-variate Shift	Available	Unavailable	Regression Classification
Unsupervised Transfer Learning		Unavailable	Unavailable	Clustering Dimensionality Reduction

Table 2.2: Settings for transfer learning given data availability (adapted from [41])

For our case, the feature space $\mathcal{X}$ and the label space $\mathcal{Y}$ are assumed to be the same between different wind farms. $P(X)$ and $f(\cdot)$ are however both different, resulting in both the domains and the tasks being different. We also have available data for both the source and target wind farms. Our use case is therefore *inductive transfer learning*, which highly relates to *multi-task learning* (MTL).

Caruana [8] describes multi-task learning as an approach to inductive transfer learning where models for multiple different tasks are trained simultaneously, all sharing knowledge and benefiting mutually from each other. The multi-task learning context fit our system needs very well. We are interested not only in applying transfer learning to improve performance on new wind farms, but also leveraging this new data to improve performance on old wind farms.

2.4 Practices and Techniques in Software Engineering

A well working machine is about more than just its functionality; it also requires certain qualities. Similarly the development of quality software requires planning and well working procedures and practices. This section will firstly define modifiability and scalability before presenting some of the common techniques that are used to develop services with high modifiability and scalability.

To discuss system qualities in regard to modifiability and scalability, the concepts must first be define for our usage. The International Organization for Standardization (ISO) has developed a standard for system quality requirements presented in ISO/IEC 25010:2011 [25]. They define modifiability as:

> Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.

Put simply we want to make it easy for TrønderEnergi to modify the system, specifically in terms of being able to support different machine learning models and also the potential of supporting other time series related problems.

Techniques used to achieve a specific quality attribute are called architecture tactics [2]. Tactics are design decisions that directly affects one or more quality attributes of a system. In the case of modifiability, Bass et al. [2] presents four tactics that can be used to improve the modifiability of the system; reduce module size, increase cohesion, reduce coupling and defer binding. In this context cohesion refers to the degree to which the responsibilities within a module are aligned, coupling refers to the reliance between modules and defer binding refers to the use of generic code that allows mapping of specific values or concepts late in the development process.

An ISO standard for scalability does not exist, so in this thesis we will use the definition by Bondi [4] who defines scalability as:

> The ability of a system to accommodate an increasing number of

> elements or objects, to process growing volumes of work gracefully, and/or to be susceptible to enlargement.

For the proposed system we are specifically interested in its scalability in terms of its ability to scale to an increase in active users.

There are two ways of achieving scalability in a system; horizontally and vertically. Horizontal scaling, also known as scaling out, refers to adding more resources to a logical unit [2], such as adding another compute instance to a cluster. Vertical scaling also known as scaling up refer to adding more resources to a physical unit [2], such as adding to the CPU or memory capacity of a single compute instance. Both types are used today, but as soon as an applications presides the capacity for a single machine, horizontal scaling is the only option.

The first thing to consider when developing a system with specific quality attribute goals, is the architecture of the solution. Traditional systems often have a large code base that does everything. Such systems are known as monolithic systems. These monolithic systems are not ideal for modifiability and scalability. Changing them require an understanding of a large and possibly complicated application, and scaling can only be done to the entire application, even if only part of it needs extra compute or memory. In this thesis we will focus on an architecture known as microservices, that offer better scalability and modifiability then monolithic systems [40].

2.4.1 Microservices

Microservices is a type of software architecture where each software module is operated as an independent software system. One of the key advantages of this approach is the ability to use different technologies in each microservice. For instance it is possible to have a web service that is using Java, and it can use Python to perform some sub task. This advantage means that it is possible to leverage the best language for each task.

Another advantage of having smaller modules is the ability to scale each module individually and independently. This is particularly useful for systems where some functionality requires extra computation power for a short amount of time, such as when training new machine learning models. This also means that each module also can be deployed to different compute targets, where the compute target is optimized for CPU or GPU workloads.

In the case of modifiability microservices makes each module smaller, and therefore makes it easier to understand and modify a single module. This has the advantage of making it easier to divide the system into smaller organizational units, and it is important as smaller teams working on smaller repositories often are more productive [40]. An example of this is that developers can work on one module, while data scientists can work on another. The small modules can also be easier to replace at a later time, as the barrier to replace modules are often lower for microservices than for monolithic systems [40]. This is particularly important as the projects grows and older modules may become outdated. Finally the small modules can be deployed separately, reducing the risk of a system wide downtime. To achieve frequent updates of the service, microservices are typically used together with DevOps practices.

2.4.2 DevOps

The word DevOps is comprised of the abbreviations for development and operations, according to Ebert et al. [16] it aims to automate the building, testing and release of software, by building a closer relationship between operations and development. DevOps is a practice that combines process with software practices and tools to allow a faster and more frequent delivery of software services, while also delivering high quality. This will make it easier to modify the system, as changes can be released with less overhead. The three terms explained below can be used to describe different stages of the deployment process.

Continuous Integration

Continuous integration(CI) is the process of integrating code into a shared repository on a frequent basis. Frequent integration allows for faster detection of integration errors, and thus less time backtracking. When using CI, it is also common to have automated tests and build scripts to ensure the code can merge without errors.

Continuous Delivery

Continuous Delivery(CD) is the process of automating delivery of software such that it can be deployed at any point in time. However the process of starting the deployment is done manually.

Continuous Deployment

Continuous Deployment(CD/CDE) is a process where software functionality is delivered frequently through automated deployments. As such it differs from continuous delivery in that no manual interaction is needed in the process.

A fully autonomous deployment utilizes continuous integration and continuous deployment, often written as CI/CD. However it is possible and sometimes also desirable to only use continuous delivery, depending on organization needs. In the case of microservices each service will have its own deployment strategy and the level of automation can be set up individually for each service. CI/CD can be setup and ran manually, however many cloud provides offer solutions that simplifies the process.

2.4.3 Cloud Computing

Cloud computing is the offer of on-demand computer system resources over the internet as a service [33]. This means users can deploy applications to the cloud without knowledge about available resources or having to purchase equipment. This reduces the upfront cost of deploying new solutions, and due to economy at scale the cost also stays low once operative. Many cloud provides also offer tools and service for CI/CD, logging, compliance, authentication, backup and more, providing a more efficient experience then on-premise solutions.

Cloud services are typically categorized into one of four architectural categories; infrastructure as a service (IaaS), platform as a service (PaaS), serverless, and software as a Service (SaaS). IaaS is a barebone solution that typically gives users access to services like Virtual Machines. PaaS are managed to a higher degree, and typically give the ability to develop, manage and run applications within a given environment. Serverless removes the need to manage infrastructure, while SaaS is the highest degree of management and offers completed, managed software.

As presented earlier in this chapter, we intend to utilize a microservice architecture. This means the system can use one or more cloud services to deploy a set of independent software modules. This can be done in many ways, but one common way is to utilize containerization. Containers can be ran in IaaS, PaaS and serverless environments and some providers also offer container native environments sometimes referred to as containers as a service (CaaS).

2.4.4 Container Technologies

Containers enable software to run in isolated environments, while offering lower overhead than traditional virtual machines [39]. Multiple containers can share the same operating system kernel, running on the container engine, while virtual machines require a full operating system for each environment managed by a hypervisor.

Docker [14] is a popular platform for containerization. The Docker platform enables developers to run their application both locally and in production using the same platform. Docker applications are configured using Dockerfiles, that specify a list of commands that will be used to create a Docker image. A Docker image consist of multiple read-only layers corresponding to the commands in the Dockerfile. Docker containers can in turn be created using Docker images.

The combination of microservices, DevOps, cloud services and container technology is specifically selected to create a system that is both modifiable and scalable. They offer a good framework to achieve the desired quality attributes, however equally important is the implementation of each microservice. One of the tactics we utilized to get a system that is highly modifiable and that applies both to the system architecture and implementation, is to reduce the technical debt of the system.

2.4.5 Technical Debt

Technical debt is the concept of choosing the quick or easy solution now, instead of a possibly harder to implement solution that will offer less work later [6]. Technical debt can be compared to monetary debt in the sense that debt accumulate interest, that will have to be repaid later. The longer the easy solution exists, the more work will be required to improve it later. The term is used to emphasize the need to make proper preparation, and complete tasks before they can cause additional issues.

Reducing technical debt is essential to ensure that a system can evolve and be maintained. According to Kruchten et al. [27] many agile teams seem to believe they are immune to technical debt. However due to time constrains and the need to deliver quick, technical debt can easily accumulate. Technical debt should not always be avoided, in some cases the short term benefits can outweigh the long term cost. In our case however, the system is intended to be delivered to TrønderEnergi as a highly modifiable system and as such reducing technical debt is vital.

Chapter 3

Related Work

For the proposed system to get the desired capabilities and ensure that it is competitive against current competition, we have conducted research into existing forecasting provides and a literature review into the current state-of-the-art methods within software engineering and machine learning. The literature review first aimed at getting a wide understanding of the topics, before diving into the concepts that were most relevant to our system. For machine learning it meant investigating time series forecasting and transfer learning, particularly transfer learning applied to wind power prediction. As for software engineering this meant requiring practical knowledge of microservices and investigating machine learning solutions in a complex software system. Based on the literature review, this chapter will present the research we think can have an impact on our system.

3.1 Current Commercial Competition

Forecasting within the energy sector and specifically for the application of wind power is nothing new. In fact, numerous commercial actors exists in the market today and TrønderEnergi has tested forecasts using the services of companies such as Alpiq [1], Powel [42] and ConWX [10]. TrønderEnergi's experience was that the processes used by these companies requires vast amount of manual labor and that the process often takes weeks to complete. This manual process makes it difficult for both the forecast providers and the power companies interested in the service.

Automating the forecast creation process does not only make the process easier for potential customers, but it also means that the marginal cost of creating new projects are only dependent on the cost of running the service. This cost advantage can for instance be used to offer free trails or offer significant savings for multiple parks. This also means that the service greatly benefits from a large customer base, meaning that another option is to offer the service at a discount compared to the competitors.

Based on our research we were able to find one company by the name of WindSim [52] that offers a similar level of automation to our proposed service. Compared to WindSim we think our proposed system offer a few advantages. Firstly for our proposed system the end user is not required to have any knowledge about wind power forecasting, WindSim on the other hand requires users to select between multiple options including artificial neural network (ANN) and computational fluid dynamics (CFD). Secondly we intend to leverage the power of transfer learning to improve forecasts as the service get more customer and more data. Thirdly the proposed system is intend to be made in such a way that also the sign up and payment for the service can be automated.

The commercial actors offering custom implementations are not the only competitors to the proposed system. Large corporations such as Amazon, Google and Microsoft offer Automated machine learning(AutoML) as part of their Cloud offer. AutoML allows users to apply machine learning to real-world problems whiteout the need for machine learning expertise [23]. AutoML automates the process of feature engineering, algorithm selection and hyperparameter optimization, meaning that no knowledge of these topics are required from the user. AutoML services are can typically be accessed using an online web interface, making them easy to use. The vast scale of the cloud providers also means they are very competitive from a price perspective. Table 3.1 illustrates the advantages of each solution.

	Proposed System	Traditional Providers	AutoML	WindSim
High forecast accuracy	X	X		X
Ease of use	X		X	X
Low marginal cost	X		X	X
Utilize transfer learning	X		X	

Table 3.1: Comparison of commercial competition

3.2 Time Series Forecasting and Transfer Learning

This section will present a selection of related works on time series forecasting and transfer learning. The aim is to give insight into the recent, most successful approaches for time series forecasting, a broad understanding of relevant transfer learning approaches and concrete examples on applications to the wind power domain.

3.2.1 The M4 Competition

The 2018 M4 forecasting competition [32] is the largest scale time series forecasting competition to date, consisting of 100,000 single-variate time series, and 50 submissions. The major findings from the M4 competition was that pure machine learning methods performed poorly, with none of them beating the simple statistical baseline of the competition. Combined methods utilizing multiple methods, mostly statistical methods, were the consistently best performing methods of the competition, with 12 out of the 17 best methods being combination methods. The two best methods, performing well for both point prediction and prediction intervals, were both hybrid methods, combining statistical and machine learning methods.

1st Place: ES-RNN

ES-RNN by Slawek Smyl [48] is the winning method of the M4 forecasting competition, combining exponential smoothing (ES) with an LSTM-based recurrent neural network (RNN). ES-RNN combines the concept of time series specific local models with the concept of cross-time series global models into a hybrid model.

Winter-Holt exponential smoothing with level and seasonality is implemented as a preprocessing step, using exponential smoothing parameters learned for each time series separately. The resulting, normalized and deseasonalized time series is then fed to an LSTM-based neural network, which is trained on all time series in the training set. The resulting time series prediction is then denormalized and reseasonalized using exponential smoothing. The globally trained RNN in ES-RNN means that transfer learning is an integral part of ES-RNN, with the network learning to make forecast on time series spanning a wide range of domains, all preprocessed for increased homogeneity.

ES-RNN is the current state of the art for single-variate time series forecasting, but is not applicable to multivariate time series forecasting. Independent exponential smoothing preprocessing of input time series in multivariate time series prediction would eliminate the relationships between variables that are the basis for good predictions. The core idea of combining learned, problem instance specific preprocessing with a global model trained across all problem instances is however a relevant approach also for multivariate time series forecasting.

2nd Place: FFORMA

FFORMA (Feature-based FORecast Model Averaging) proposed by Montero-Manso et al. [38], is the second place winner method of the M4 forecasting competition. As with ES-RNN, FFORMA combines local, statistical models with a global machine learning model, utilizing transfer learning. FFORMA takes a meta-learning and meta-transfer approach, where a machine learning model is trained to combine different statistical methods.

FFORMA consists of an ensemble of multiple different, statistical forecasting models, such as exponential smoothing and ARIMA. These models are fitted to each time series individually. Rather than combining these different forecasts statically using fixed weight, a machine learning based meta-model is trained to predict the weight of each individual forecasting method based on the specific time series. The meta-model uses the gradient tree boosting implementation XGBoost. Input to the meta-learner is a set of 42 features extracted from the time series, such as strength of trend, strength of seasonality, etc.

FFORMA illustrates an interesting meta-transfer approach that is also applicable to multivariate time series. Training a global model to combine results of multiple local models. A limitation is however the potentially large amount of problem instances required to train the meta-model. Since each problem instance only gives a single training example for training the meta-model. While 100,000 different problem instances were available in the M4 competition, such instance availability is not realistic for most problem specific systems within the energy sector, and certainly not for wind power production.

3.2.2 Deep Transfer Learning

In a survey by Tan et al. [49], reviews the current work on transfer learning on deep neural networks, and specify four main categories of transfer learning within deep learning: *instance-based*, *mapping-based*, *network-based* and *adversarial-based* methods.

Instance-based

Instance-based approaches are based on the selective reuse and weighting of some instances in the source domain to augment the training set for the target domain.

Source domain examples are weighted based of similarity to target domain, with more similar source examples being weighted higher. Instance-based methods assume that there is a partial overlap between the source data and target data within feature space with similar label mapping.

Mapping-based

Mapping-based approaches attempt to increase homogeneity between target and source data by mapping the data to a shared, intermediate space. Mapping can either be applied as a preprocessing step, with unique mapping functions for each domain. The simplest example of this would be to normalize data for each domain independently. Alternatively, adaption layers can be worked directly into the network using domain confusion [51].

Network-based

Network-based approaches are based on reusing parts of neural networks trained on source domains to create and train a neural network for the target domain. Neural networks consists of multiple layers, where each layer represents a transformation through different internal representations. Particularly early layers in neural networks learn to extract general, useful features from the input data. Network-based approaches assume that some internal representations learned on the source domain can also be useful on the target domain.

Adversarial-based

Adversarial-based approaches utilizes an adversarial learning process to improve generalization of internal representations learned within a network between source and target. A neural network with shared early layers between source and target, but separate output layers, is trained on source and target data simultaneously. In addition to the label prediction, a domain classification output layer is added, which is trained to classify whether the example originates from the target domain or the source domain. See Figure 3.1 for example layout. The inverse of the domain classification loss is worked into the loss function for the entire network, meaning the network is optimized to minimize prediction error while maximizing domain classification error simultaneously. The addition of this adversarial component drives the learning of the shared layers to learn better shared representations for the source and target domain compared to normal network based multi-task learning methods, such as presented in subsection 3.2.3.

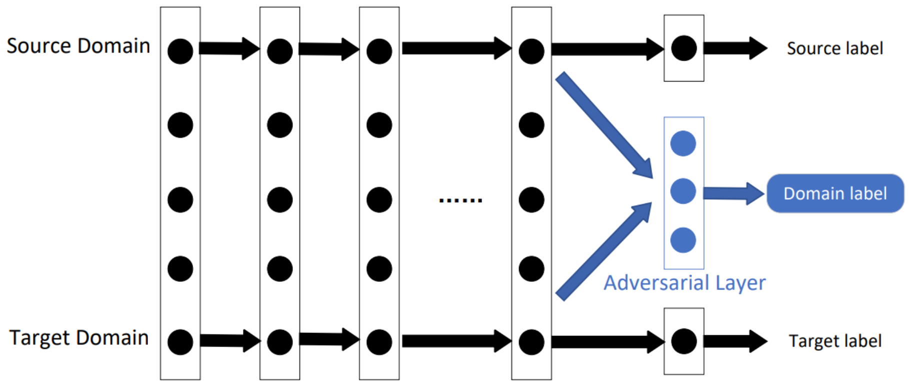


Figure 3.1: Example neural network with adversarial domain classification label. (adapted from [49])

3.2.3 Wind Power Forecasting: Multi-Task Neural Network

Hu et al. [22] experimented with a neural network based multi-task learning (MTL) approach for short-term wind speed prediction. The experiments were performed on wind speeds measured at different wind farms, and not the wind farm power production itself. Experiments were performed on data from four different wind farms spaced out across northern China, with distinct differences in climate and topology.

The model is a multi-task learning based neural network, with a set of shared hidden layers and separate output layers for each individual farm. The model consists of two hidden layers with 100 units each. The hidden layers were pre-trained unsupervised using auto-encoders, before supervised fine-tuning with labeled data. The output layer for each farm and the hidden layer fine-tuning is done in parallel for all farms, ensuring that the hidden layers learn a representation effective for forecasting on all farms. See Figure 3.2 for an illustration of the network structure.

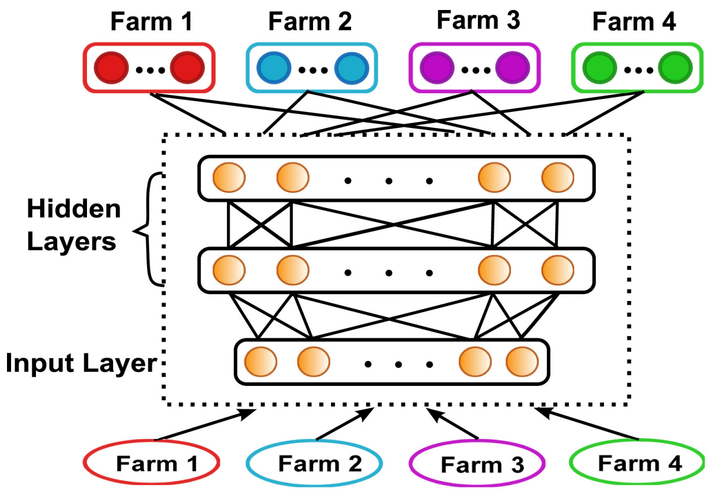


Figure 3.2: The MTL network for wind speed forecasting used by Hu et al. [22] (adapted from [22])

Data with 10-minute data is used, with input to the model being wind speed for the past 24 hours (144 timesteps). The model is used to predict future wind speeds multiple timesteps ahead. The strategy for multi-step forecasting is not specified, but the multiple output neurons per farm in Figure 3.2 indicates a multi-output strategy, with one output neuron for each predicted timestep. Experiments were conducted with transfer to one target farm with different, smaller quantities of data from the three other farms as sources, with one year of data each. Results are compared to several non-transfer methods trained only on the target wind farm data. A neural network with the same architecture as the transfer model, but with output only for the target wind farm, a support vector regressor with linear kernel and an extreme learning machine with linear kernel. Their experiments show consistent good results with the MTL network, particularly for shorter forecasting horizons and smaller target data availability.

While these experiments show multi-task learning of neural networks with shared layers to be a promising strategy for wind power forecasting transfer learning, there are several limitations with these experiments that make them less relevant for our work. Most importantly, the experiments are based of single-variate time series forecasting, using only historical values of the time series to be predicted, and does not include weather forecasts. The experiments were also performed on wind speed measured at the wind farm, not actual wind farm power production. While experiments for seven different quantities of target data were performed

(0.5 month and 1-6 months), we find this granularity to be insufficient. Results were also only reported for 10-minute, 30-minute, 1-hour and 2-hours forecasting, while we wish to make forecasts for several hours into the future.

3.2.4 Wind Power Forecasting: DNN Ensemble Meta-regression

DNN-MRT

Qureshi et al. [45] introduces a neural network based ensemble for wind power forecasting called DNN-MRT (Deep Neural Network based Meta Regression and Transfer Learning). DNN-MRT utilizes a neural network based meta-regressor to combine results from an ensemble of neural network based base regressors, and utilizes transfer learning by re-using pre-trained base regressors from one wind farm to speed up the learning process. The model uses previously generated wind power and weather forecasts as inputs.

The model consists of an ensemble of nine base regressors, each being a deep neural network. In addition to the forecasting loss, the base regressors utilize L2 weight regularization, which optimizes weights to be small, and sparsity regularization, which optimizes average output of each neuron within the network to be small to enforce output sparsity. The base regressors are pre-trained as stacked autoencoders, before they are fine-tuned with labeled data. Each of the nine base regressors consists of different network structures and hyperparameter choices, to facilitate variety in base regressor representations. The forecasts from each base regressor is combined with the original input and used to train a meta-regressor for final forecast, which is a deep belief network.

DNN-MRT utilizes transfer learning to speed up the learning process of base regressors. The base regressors are only pre-trained on the first wind farm, and models for subsequent wind farms reuse and fine-tune these pre-trained base regressors. The result is a faster training process for new wind farms, but does not leverage transfer learning to improve performance on target wind farms.

DNN-MRT was tested for single-step forecasting on five different wind farms and compared to the performance of three other relevant works on wind power forecasting. DNN-MRT was found to give better performance than previous works, particularly in terms of forecast stability, as seen by a greater improvement in RMSE compared to other techniques than MAE.

DNN-MRT shows positive effects of neural network ensemble techniques for wind power forecasting, which is a promising modelling approach given neural networks flexibility for transfer learning techniques. DNN-MRT's use of transfer learning is quite limited, only focusing on training time reduction and not on model accuracy improvement for smaller data quantities. Additionally, no experiments were conducted to actually evaluate the impacts of transfer learning on training time.

ATL-DNN

Qureshi and Khan [44] expands on their previous work with DNN-MRT by introducing ATL-DNN (Adaptive Transfer Learning in Deep Neural Networks). ATL-DNN functions similarly to DNN-MRT, but this work focuses on designing and online wind power forecasting system to handle the continuous income of new data for all wind farms.

Similar to DNN-MRT, ATL-DNN utilizes an ensemble of neural network base regressors pre-trained as stacked, sparse auto-encoders, and a deep belief network as a meta-regressor to make a final prediction. The meta-regressor takes the original input and the base regressor predictions as input to make a final prediction. In ATL-DNN, rather than having an ensemble of nine base regressors with different parameters, the ensemble consists of three base regressors with the same parameters, but trained on different subsets of data. Each of the three base regressors for all wind farms are fine-tuned on a base-regressor pre-trained on the first four months of data from the same, randomly chosen wind farm. The base regressor are trained on the first four, eight and twelve months of training data respectively. An online training scheme for training a new base-regressor to replace the oldest base-regressor every four months is then proposed. This new base regressor is trained on the data for the previous base regressor plus the four next months of data. Figure 3.3 shows the training and testing procedure of the system.

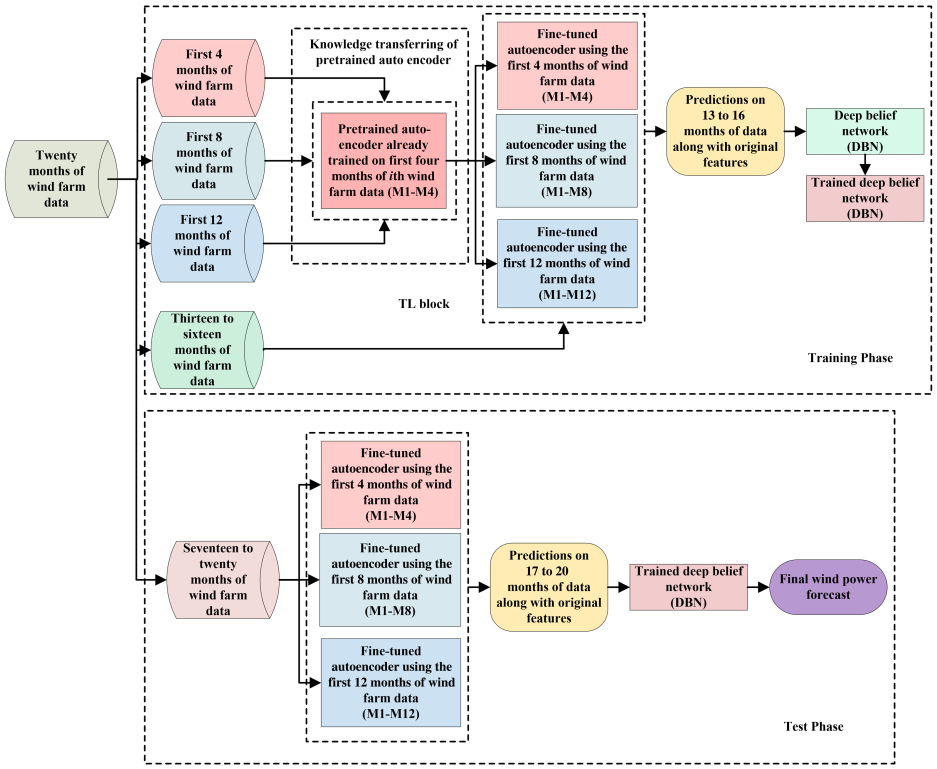

Twenty months of wind farm data
First 4 months of wind farm data
First 8 months of wind farm data
First 12 months of wind farm data
Thirteen to sixteen months of wind farm data
Knowledge transferring of pretrained auto encoder
Pretrained auto-encoder already trained on first four months of ith wind farm data (M1-M4)
TL block
Fine-tuned autoencoder using the first 4 months of wind farm data (M1-M4)
Fine-tuned autoencoder using the first 8 months of wind farm data (M1-M8)
Fine-tuned autoencoder using the first 12 months of wind farm data (M1-M12)
Predictions on 13 to 16 months of data along with original features
Deep belief network (DBN)
Trained deep belief network (DBN)
Training Phase
Seventeen to twenty months of wind farm data
Fine-tuned autoencoder using the first 4 months of wind farm data (M1-M4)
Fine-tuned autoencoder using the first 8 months of wind farm data (M1-M8)
Fine-tuned autoencoder using the first 12 months of wind farm data (M1-M12)
Predictions on 17 to 20 months of data along with original features
Trained deep belief network (DBN)
Final wind power forecast
Test Phase

Figure 3.3: The training and testing procedure for ATL-DNN (adapted from [44])

ATL-DNN is tested on five different wind farms, using one as source for pre-training the base regressor. The meta-regressor consistently outperforms all base regressors, showing the positive value in an ensemble of base regressors trained on different data subsets. The performance of the system is compared to four other, relevant works on wind power production, including DNN-MRT, achieving the best performance.

ATL-DNN, as with DNN-MRT utilizes transfer learning to speed up the training process, not to improve model performance. ATL-DNN proposes an online learning approach where only one new base regressor needs to be retrained per four months, rather than retraining the entire system. The concept of online learning is relevant for our system in terms of better leveraging newer data and handle potential seasonal drifts within the data, but we believe four months is too long for this purpose, and a more regular retraining strategy would be more effective. Experiments were not conducted to evaluate the effects of the online retraining procedure beyond the first set of base regressors illustrated in Figure 3.3.

3.2.5 Wind Power Forecasting: Instance-based Transfer Learning with Gradient Boosting Decision Trees

Cai et al. [7] tests an instance-based transfer learning method with gradient tree boosting, which they call IBT-GBDT (Instance-Based Transfer Gradient Boosted Decision Trees) for wind power quantile regression. The method uses weighted source wind farm data to augment the training set for the target wind farm, using weights generated by analyzing error distribution in an iterative process.

In IBT-GBDT the weights for the target wind farm is a fixed hyperparameter, and weights for source wind farms are initialized to 1. The source weights are then refined iteratively by first training a gradient tree boosting model with the current weights, and then using predictions on each source wind farm to recompute weights based on the error distribution. The error distribution is assumed to be Laplace distributed, and maximum likelihood is used to estimate the scale parameter of the Laplace distribution for each source wind farms based on predictions. The source weights are then re-computed as the multiplicative inverse of the scale parameter for the source. All source weights are then scaled by dividing by the smallest source weight. This process is repeated with the new set of source weights, until convergence.

Experiments showed that IBT-GBDT achieves notable improvements compared to training gradient tree boosting only on data for the target wind farm, and significantly better than training on a combined dataset of target and source data with equal weights. The model performs particularly good compared to alternatives when there is little data available for the target wind farm.

3.2.6 Wind Power Forecasting: Cluster-based Predictor Weighting

Tasnim et al. [50] introduces a cluster-based similarity estimation approach to generate weights for an ensemble of predictors trained on different sources for day-ahead power forecasting on a new target wind farm. Only historical wind data is utilized as input to the model, without historical power production or future weather forecasts, with power production being the prediction target. Because historical wind data is available in large quantities for newly built wind farms through meteorological services. This turns the problem into a transductive transfer problem, where large quantities of labeled data is available in source domains, and large quantities of unlabeled data are available in target domain. Such a source model weighting approach is known as multi-source domain adaption (MSDA), with other, instance-based (rather than cluster-based) weighting approaches existing in the literature.

The proposed cluster-based weighting strategy utilizes the similarity of clusters within the target data and source data to estimate domain similarity. For each source domain, k-means clustering is applied to generate k clusters. The k cluster centers from the source data is also applied to the target data, with each target example being allocated to the closest cluster center. Two statistical difference measures between source and target are computed for each cluster: Difference in marginal probability distribution, which is the proportion of data assigned to the cluster, and difference in conditional probability distribution, which is the distribution of label values within each cluster. Since labels are not available for target domain, predictions generated by the source predictor are used instead. These two differences across all clusters are aggregated, and multiplicatively inversed to form the relative source weight. This procedure is done for each source domain independently, and final weights are then normalized to sum to 1 across source domains.

13 different spectrum-based statistical features are extracted from the a 30-day window of past wind speed is used as input features. These features provide a more stable basis for similarity estimation than raw wind speed values. Support vector regressors trained on each source dataset are used as base regressors. The system is tested on a set of 70 datasets with synthetic labels, for locations spread across Australia. The synthetic labels are generated from transforming measured wind speed through a power curve, with the same power curve being used for all datasets. Performance is measured using each dataset as target in turn, with the other 69 datasets as sources, and averaged across all targets. The proposed method is compared to a regressor trained on all source datasets combined, an ensemble of source regressors with equal weights and an ensemble of source models with weights generated by an instance-based MSDA approach. The proposed cluster-based MSDA approach outperforms the combined dataset regressor by 12.54%, the equal weight ensemble by 16.88% and the instance-based MSDA approach by 20.63%.

The most interesting takeaway from this work is the idea of utilizing the larger amount of historical wind data, and possibly also historical weather forecasts available for before the construction of a new wind farm to improve transfer from other wind farms. This approach utilizes no power production data for target farms, allowing potentially good predictive models to be constructed earlier for new wind farm than what is possible with target wind farm label reliant transfer methods. An advantage with the proposed approach, is that the cluster-based weight generation approach, which can only utilize data available for before wind farm construction, is independent from the base regressor modelling approach. The result is a lot of flexibility for the base regressor modelling, which can use any modelling method and input features, including power production, which are not available historically. This approach can also be combined with other label-based model refinement strategies to improve the performance of base regressors with labeled target farm data as it becomes available.

3.3 Software Engineering for Machine Learning

This section will present how software engineering practices can be used to create machine learning models that are more reliable and easier to modify. This is done by identifying and paying down technical debt.

3.3.1 Technical Debt

This section will focus on technical debt risk factors that are specific to machine learning solutions. Sculley et al. [47] argue that while machine learning packages have the same basic code complexity as normal code, it also has a larger system-level complexity that can create hidden debt. Sculley et al. [47] then goes on to introduce some of the common reasons for hidden technical debt in machine learning.

Eroded Boundaries

Software engineering practices has demonstrated that enforcing boundaries using encapsulation and modular design typically help create maintainable code [47]. However it is hard to enforce the same boundaries for machine learning solutions due to its entanglement to the underlying data, as a consequence changing anything changes everything(CACE). In return this makes it difficult to modify existing machine learning models, due to the uncertain outcome.

Hidden feedback loops can lead to issues in analyzing system performance [47]. A hidden feedback loop arises when the performance of the system itself changes the prediction problem. This can lead to an additional drift in the distribution not present in historical data. Quick experiments before system deployment would not capture these feedback loops, making analyzing the system difficult.

Predictions made from machine learning models are often consumed by other systems. Without access control to the predictions, it is possible to have undeclared consumer that uses the prediction as input. In such a case changing the machine learning model can cause issues for its undeclared consumers and as a consequence making it difficult to change the machine learning model at all.

Data Dependencies

Kruchten et al. [28] argue that dependency debt contributes to code complexity and technical debt in software engineering. Based on this Sculley et al. [47] argue that data dependencies can build debt in a similar manner for machine learning systems. For instance some input features are unstable, meaning they can change behavior over time. This can cause unexpected behavior in a machine learning model due to CACE. One strategy to prevent unstable data dependencies is to use version control of the data, and migrate to the newest version when it is fully validated for the input system.

Underutilized data dependencies add complexity to the system, as it makes the system vulnerable to changes. Therefore efforts should be made to frequently evaluate the need for every feature, and remove any feature that does not provide sufficient value.

System-level Spaghetti

One common problem for machine learning solutions is the extensive use of the glue code design pattern, in which high amounts of supporting code is written to get data in and out of libraries. This can make it difficult to test other packages or other machine learning approaches. In the case that the modules are written in different languages it may be beneficial to re-implement specific algorithms within the system, reducing glue code and making the system easier to maintain and test.

A special case of glue code known as pipeline jungles can appear if care is not taken when adding and merging new data sources. In worst case the resulting system can become a jungle of scrapes, joints and sampling steps with intermediate file outputs, making the system difficult to test and maintain.

Finally it can be tempting to perform experiments using conditional branches within the main production code. This can however cause issues over time and it is recommend to periodically examine each experimental branch to see if it can be removed.

Usage for Our System

One of the system requirements is that the system should be modifiable. Systems that accumulate a high degree of technical debt, can be difficult to modify, as such our aim was to reduce technical debt. Due to the unpredictable nature of machine learning solutions, extra care should be taken to ensure the quality of the machine learning solution.

3.3.2 Machine Learning Production Readiness and Technical Debt Reduction

When preparing new software for production, it is vital that it is well tested and works reliably. Breck et al. [5] presents 28 specific tests and monitoring needs to improve production readiness and pay down machine learning debt. Although testing seems obvious, in a survey of a dozen projects at Google, none of the suggested tests where implemented by over 80% of the teams.

Tests for Features and Data

Unlike traditional systems that are only dependet on the source code, machine learning systems behavior is also dependent on the input data. Therefore Breck et al. [5] proposes a set of tests to validate the input data, this include tests for feature expectations, feature benefits, feature cost, meta-level requirements and time to add new features.

Tests for Model Development

Breck et al. [5] propose a set of tests to ensure quality models, this includes the use of code review, comparing offline and online proxy metrics, tuning of all hyperparameters, impact of model staleness, verify that simpler models are not better, ensuring that small improvements in overall model performance don't significantly impacts subsets and ensure model fairness.

Tests for ML Infrastructure

Breck et al. [5] have created a set of tests for the project infrastructure, including the project pipeline. The tests consist of unit, integration and canary testing, as well as testing for step by step debugging, reproducibility, model validation and rollback capability.

Monitoring Tests for ML

The tests for monitoring were created to ensure reliable operation after a product is launched. The tests includes the use of notifications in case of changes to dependencies, notifications if the input data differ significantly from expected value, ensure that the system produces the same values in training and production, monitor model staleness, validating the the model is numerically stable and monitor performance.

The ML Test Score

Based on the presented tests, Breck et al. [5] propose to validate system readiness using a test score. The test score can be used as a guideline by developers to improve system readiness and as such improve reliability. The test score is calculated by first finding the score for each category, this is done by adding the points for each test in the category. For each test half a point is rewarded if the test is conducted manually and one point if conducted automatically. The final score is the minimum score of the four categories, and it can be used to validate system readiness using Table 3.2.

Points	Description
0	More of a research project than a productionized system.
(0,1]	Not totally untested, but it is worth considering the possibility of serious holes in reliability.
(1,2]	There's been first pass at basic productionization, but additional investment may be needed.
(2,3]	Reasonably tested, but it's possible that more of those tests and procedures may be automated.
(3,5]	Strong levels of automated testing and monitoring, appropriate for mission-critical systems.
>5	Exceptional levels of automated testing and monitoring.

Table 3.2: Interpreting an ML test score (adapted from table V in [5])

Usage for our system

Sculley et al. [47] presented a set of technical debt risk factors, while Breck et al. [5] presents a framework that can help reduce technical debt and improve system reliability. This framework can be use in conjunction with the presented risk factors to help reduce technical dept. Even without implementing the framework it can be used as a steeping stones toward a reliable commercial system.

3.4 Software Architecture

This section will present the literature that is relevant to the overall design of the system. As the system is intended to be developed using microservices, this will serve as the main focus of this research. First the section will present how architecture technical debt can be reduced in a microservice architecture, such that the system is more maintainable and easier to modify over time. Second it will investigate the possibility of using a serverless platform with microservices, as a solution for very unstable work loads.

3.4.1 Architectural Technical Debt in Microservices

de Toledo et al. [11] states that microservices is a popular way to achieve continuous delivery, but that it also can create architecture technical debt. de Toledo et al. [11] conducted a case study into the microservices of a large financial institution, with over 1000 microservices. Based on this research they where able to find specific issues, that could be useful to consider in the design of our system.

According to Kruchten et al. [27] architecture technical debt is the most challenging type of technical debt to uncover. Therefore having awareness of possible architecture technical debt and the cost of refactor can reduce it in the first place. Based on interviews de Toledo et al. [11] came up with a list of issues and list of solutions based on the company's experience.

The main issues related to architecture technical debt was found in the communication layer, which is the way microservices communicate. de Toledo et al. [11] found a high number of point-to-point communications among services, meaning that the services had different interfaces for each service they communicate with. This increases the coupling between the services and is the opposite of the intended use of microservices, which is to reduce coupling. In turn this makes it difficult to maintain the system, as every data connection must be maintained individually. This can however be fixed by rewriting the point-to-point connections to one common connection point.

Another issue is the existence of business logic in the communication layer. de Toledo et al. [11] were able to discover cases where data was transformed between endpoints. In addition the company being researched had a policy that required that changes first had to be agreed upon by the communication layer team. This significantly slows down the development process, and can also result in new business logic being added with new endpoints. This case can be avoided by moving the business logic from the communication layer, to the services themselves.

A third issue is that the teams creating the microservices had not agreed upon format, as a result many services needs to transform and filter their data to communicate. This issue can be reduced by defining a canonical model per domain, that is to create a standardized model.

The paper clearly identified the need for standardizing communication among microservices. If such issues are not addressed, they can cause significant interest on the technical debt. However with planning technical debt can be decreased, making the system easier to modify and maintain.

3.4.2 Serverless Computing for Container-based Architectures

In resent years serverless computing has emerged as a fully managed alternative to IaaS and PaaS. A major advantage of serverless, is that it is event driven and only runs for the execution time of the program. In addition due to its manged nature, scaling can be done without the need for extensive configuration. However most providers only offer serverless in specific programming languages and environments. Therefore Pérez et al. [43] propose a framework that can run Docker images on top of serverless applications and thereby removing the environment restriction.

The framework is intended to be used for highly-parallel event driven applications. Pérez et al. [43] tests the framework using an image path recognition framework based on deep learning. By running AWS lambda (serverless offer by AWS), they were able to find 4 575 objects from 1 000 images in 2 minutes and at a cost of less than one US dollar. The use of serverless for making predictions in deep learning solutions seems promising, and could be an important tool to handle high number of requests at specific times, and reduce the overall system cost.

As the tests performed by Pérez et al. [43] are similar to the predictions we will be making for our system, it presents an opportunity to explore serverless computing, as a possible platform.

3.5 Summary

Based on the presented literature, we have gained a better understanding of the possible methods and techniques that can be used to created a scalable machine learning service. We see several relevant applications of transfer learning in state-of-the-art single-variate time series forecasting through the M4 competition, and successful application on transfer learning for improved prediction accuracy for wind power forecasting. Few works have been published on the use of transfer learning for wind power forecasting. However, the presented works on wind power forecasting shows promising effects of transfer learning on performance in an MTL network setting as presented in subsection 3.2.3. This technique can be further expanded with adversarial techniques as presented in subsection 3.2.2. subsection 3.2.5 shows that experiments with transfer learning for wind power forecasting using gradient tree boosting has been done previously, using instance-based transfer learning. We could however not find any examples of model-based transfer learning approaches with gradient tree boosting being applied to wind power forecasting. The idea of leveraging the larger amount of available historical wind data without power production data, as was done by the cluster-based MSDA presented in subsection 3.2.6, is also worth exploring further for our system.

We were also able to find relevant literature related to software engineering that could help improve the qualities of the system. section 3.3 presents the concept of hidden technical debt in machine learning systems, and a framework to reduce technical debt and improve production readiness. The investigation into technical debt for microservices, discovered concrete cases that can be used to improve the modifiability of our system. The research into serverless computing shows potential for running containerized application in a highly scalable environment. By utilizing the work done by other researchers, we have the potential to create a system with low technical debt, that is highly scalable, reliable and maintainable.

Chapter 4

Forecasting and Transfer Learning Methods

4.1 Transfer Learning for Gradient Tree Boosting

In TrønderEnergi's previous experiments with wind power forecasting, Gradient Tree Boosting (GTB) has been the machine learning method yielding most success. In addition to good end performance, GTB has few parameters that require tuning, and has achieved consistently good results on wind farms without wind farm specific parameter tuning. This out-of-the-box flexibility makes it a good fit for a large scale machine learning service, where models for a wide range of different wind farms from different customers need to be trained. We therefore choose to base our transfer learning methods and experiments on GTB.

Based on the general transfer learning paradigm of source pre-training and target refinement we propose an extension to the GTB training process to include transfer learning, which we call TL-GTB in this report. TL-GTB consists of four steps:

1. **Source pre-training:** A standard GTB ensemble with n_s trees is trained on source training data.
2. **Sub-ensemble target testing:** For each integer cut-off point k where $1 \leq k \leq n_s$, test the sub-ensemble consistent of trees $\{1, ..., k\}$ on target training data.

3. **Ensemble pruning:** The GTB model is pruned to the best sub-ensemble found in in step two, discarding later trees in the ensemble.
4. **Target refinement:** New trees are learned from target training data and added to the ensemble.

Figure 4.1 illustrates the training steps of a TL-GTB model.

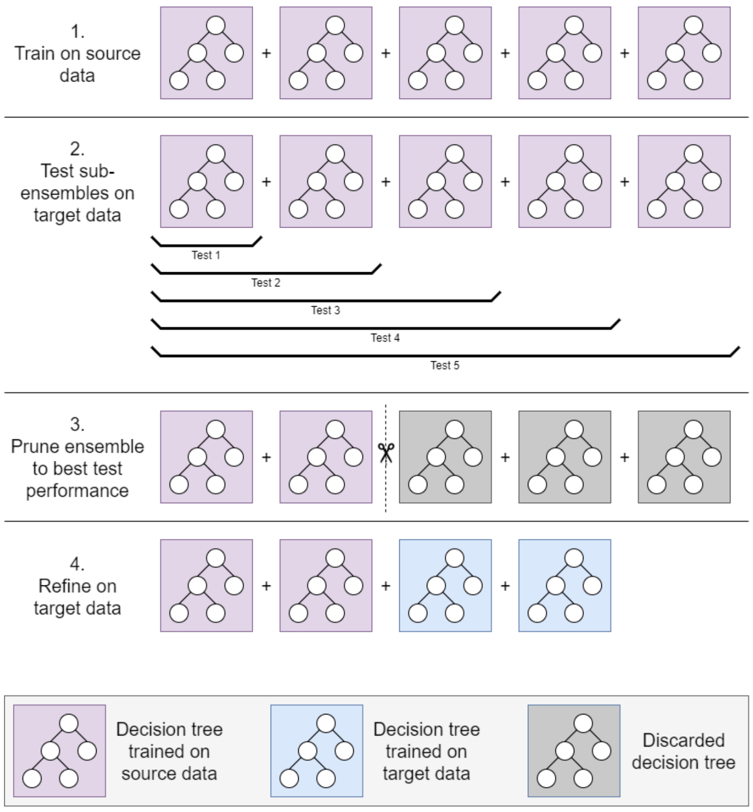

1.
Train on source data
2.
Test sub-ensembles on target data
Test 1
Test 2
Test 3
Test 4
Test 5
3.
Prune ensemble to best test performance
4.
Refine on target data
Decision tree trained on source data
Decision tree trained on target data
Discarded decision tree

Figure 4.1: Illustration of the 4 training steps for a TL-GTB model

4.2 Source Utilization Strategies

While the TL-GTB method explained in the previous section is made to transfer knowledge from one source data set, each target wind farm will have a multitude of source wind farms available as candidate source wind farms. Here we present a few alternative approaches for utilizing multiple source wind farms.

4.2.1 Single-Source Transfer

The simplest approach is to choose only one of the candidate source wind farms as source wind farm for transfer learning. Several strategies for choosing which candidate source wind farm to use could be employed, for example: Using the wind farm with most available data, the wind farm with shortest geographical distance to the target wind farm, the wind farm with best initial performance on the target training set after step two of the TL-GTB training process or the wind farm with greatest weight following one of the weighting strategies in section 4.3.

4.2.2 Combined-Source Transfer

An alternative that utilizes data from multiple or all candidate source wind farms, is to combine the training sets of all source wind farms into one, and using this combined dataset as the source training set for TL-GTB. This strategy is able to utilize a larger amount of data, and would be forced to learn an optimal compromise between the different source wind farms. The resulting model would focus on features and relationships that apply to all the source wind farms, and would therefore be more likely to generalize well to the target wind farm. The drawback to this strategy is however that in the case where some, but not all wind farms share more unique patterns with the target wind farm, the learning of these patterns might be lost in favor of a weaker, more generic forecaster.

Using this strategy, source wind farms can be weighted by weighting the source training data during source training. The impact of errors for each data point is multiplied with its weight, allowing the learner to prioritize minimizing error on data from higher weighted source wind farms.

4.2.3 Transfer Ensemble

A compromise between the two previous strategies, which could potentially yield the best of both, would be building an ensemble of single-source models, one for each source wind farm. The forecasts from each single-source model would be combined through an average to give a final forecast for the target wind farm. This strategy would allow each base model to learn more specific patterns for their source wind farm than the combined dataset strategy. The test and pruning steps of TL-GTB would allow selection of how much of these wind farm specific patterns to retain for target wind farm forecasting on a case by case basis. The final averaging of forecasts helps stabilizing forecasts and potentially helps improve generalization to new wind farms.

With this strategy, source wind farms can be weighted by making the final forecast a weighted average of the different single-source models, utilizing the weight of the corresponding wind farm.

4.3 Source Weight Generation

While for each new target wind farm a larger set of candidate source wind farms are available, the similarity to the target wind farm might vary greatly due to for example different location climates, weather trends and different wind turbines. We investigated different methods for generating weights for source wind farms, source wind farms more similar to the target wind farms are given greater weights, allowing more knowledge to be transferred from more similar wind farms.

4.3.1 Wind Profile Similarity-Based Weights

Inspired by the work of Tasnim et al. [50], described in subsection 3.2.6, we implement and test a cluster based wind profile similarity estimator for weight generation. The main advantage of this generation method is that it utilizes historic wind data alone, without historic production data, to estimate similarity. Such historic wind data is available from weather data providers. Therefore weights can be generated based on larger quantities of data, even for completely new wind farms with little or no available production data. This method however only generates weights based on wind profile similarity, ignoring other factors that might affect wind farm similarity, such as physical hardware.

For one source wind farm, the similarity of the source wind farm's wind profile to the target wind farm's wind profile is estimated by comparing clustering of data points. K-means clustering is used to generate a set of k cluster centers $\{c_1, c_2, ...c_k\}$, and a cluster assignment function:

$$h(x) = \arg\min_{c_a} (e(x, c_a))$$

Where x is a data point and $e(x, c_a)$ is the euclidean distance between data point x and cluster center c_a.

Given a set of data $X = \{x_1, x_2, ...x_n\}$, we define the cluster counting function for cluster a as:

$$C_a(X) = \sum_{x \in X} [h(x) = c_a]$$

Which is expressed using Iverson bracket notation, where $[P]$ is 1 if proposition P is *true* and 0 otherwise.

Given a set of source data X_s of size n_s and a set of training data X_t of size n_t we then define the wind profile difference as the sum of absolute differences in proportion of the dataset assigned to each cluster for the source dataset and the training dataset.

$$d = \sum_{a=1}^{k} \left| \frac{C_a(X_t)}{n_t} - \frac{C_a(X_s)}{n_s} \right|$$

This difference is then multiplicatively inverted to get the source wind farm weight:

$$w = \frac{1}{d+s}$$

Where s is a small, stabilizing constant, in order to ensure that wind farms with very small differences in wind profiles do not over-dominate other wind farms. This is particularly important in the case where multiple wind farms close to each other might use the exact same weather forecasts, to prevent a division by zero alternatively infinitely high weight for some source wind farms.

This procedure is performed for each of the n source wind farm, to generate a set of weights $\{w_1, w_2, ..., w_n\}$. This set of weights is then normalized in order to sum to 1, yielding the final set of normalized weights $\{\hat{w}_1, \hat{w}_2, ..., \hat{w}_n\}$ where the normalized weight for wind farm b is:

$$\hat{w}_b = \frac{w_b}{\sum_{i=1}^{n} w_i}$$

This method of weight generation leaves the number of clusters k and the stability constant s as tunable parameters.

4.3.2 Single-Source Transfer Performance-Based Weights

We also propose a transfer performance based weighting strategy, in which the error on the target training set after pruning the TL-GTB model is used to generate weights. This method allows the transferred knowledge from source wind farms to directly justify the source weights, encompassing any element of relevant similarity, such as wind profile and hardware, with an automatically balanced focus based on forecasting performance.

For each of the n source wind farms, a TL-GTB model is trained through the first two steps of the training process described in section 4.1, in order to get a set of target training set errors $\{e_1, e_2, ..., e_n\}$. A list of wind farms is then ordered on these errors from lowest to highest and used to generate a set of ordinal numbers $\{o_1, o_2, ...o_n\}$. o_k equals the position of the k'th source wind farm in the aforementioned list, 1 for the best performing source wind farm, and n for the poorest performing source wind farm. A set of exponentially decaying weights $\{w_1, w_2, ..., w_n\}$ is then generated as follows:

$$w_k = d^{o_k - 1}$$

Where $0 \leq d \leq 1$ is the exponential decay factor.

These weights are then normalized in order to sum to 1, resulting in the set of normalized weights $\{\hat{w}_1, \hat{w}_2, ..., \hat{w}_n\}$ where the weight for wind farm k is:

$$\hat{w}_k = \frac{w_k}{\sum_{i=1}^{n} w_i}$$

This weight generation method leaves the exponential decay factor d as a tunable parameter. Adjusting d allows for adjusting the balance between prioritizing the best source wind farms, versus utilizing knowledge from many wind farms. In the extreme case where $d = 0$, the resulting forecaster would be identical to the single-source model with the source wind farm selected based on TL-GTB pre-fine tuning target training error. In the other extreme case where $d = 1$, this method would generate equal weights for all source wind farms.

4.4 Multi-Step Forecasting Strategies

It is not only desirable to provide forecasts for a single hour forwards in time, but also for multiple time steps or forecasting horizons. We explore two different methods of generating forecasts for multiple time steps forwards in time.

4.4.1 Recursive Model Forecasting

The recursive forecasting method involves generating multiple forecasts with a single-step forecaster, where the results of the previous forecasts is fed back into the model as inputs. For time series data this strategy is possible when values needed for future forecasting for all input variables are known, except the target variable. In our case, input variables are past production and variables related to weather forecasts, where these weather forecasts are available for a longer time into the future. By inserting the single-step production forecast as an assumed real production, we can move the input window one step forwards and make a forecast for the 2-hour horizon. This procedure can be repeated for as long as the necessary weather forecasts are available.

This method of generating multi-step forecasts is able to utilize past production features to improve forecasts, also for larger horizons. There is however a risk of forecasting errors from earlier forecasting steps propagating and accumulating through later forecasts, resulting in unstable and inaccurate forecasts for large horizons. Due to the gradual reduction in auto-correlation of wind power production as horizons grow, combined with this error accumulation, we expect this strategy to be more effective at smaller horizons, but perform poorly at larger horizons.

4.4.2 Productionless Direct Forecasting

Another approach to forecasting for multiple horizons is to create a single model that is capable of forecasting for any horizon, by excluding past production features from the models input and only utilizing weather forecasts. Since weather forecasts are available for a longer time into the future, this allows us to generate forecasts for any horizon simply by feeding the correct window of weather forecast data is input to the model.

This multi-step forecast method results in equal forecasting accuracy for every horizon, at the cost of not being able to utilize past production as a feature. Due to the relative importance of past production for smaller horizons, we expect this to be a strategy primarily effective at larger horizons.

Chapter 5

System functionality and implementation

The primary artifact of our research is a product in form of a software system. This chapter will present the features, implementation, platform and architecture of the system in depth from a technical perspective.

5.1 Requirements

The software system is intended to be sold to other energy producers as a service that fully automates the process of wind power forecasting. This requires that the system is competitive against the current providers of similar services. As discussed in section 3.1 similar services exist, and the product is specifically meant to compete against AutoML and task specific implementations such as Alpiq [1]. To be competitive the system should ideally have the same or a greater level of automation then AutoML as a service, but with a performance that is similar to task specific implementations.

In collaboration with TrønderEnergi we have created a set of requirements that we think are essential for the system to be competitive. The requirements are created to describe both the functionality and non-functional requirements of the system. The requirements for functionality of the system is presented in subsection 5.1.1, while the non-functional requirements or qualities of the system are outlined in subsection 5.1.2.

5.1.1 Functional Requirements

To quantify the needed functionality of the software system, a set of functional requirements has been created using a iterative process. The requirements are presented in the terms of users and system administrators. A user is considered to be a customer using the service to make forecasts, while system administrators are TrønderEnergi's technical staff. A list of the requirements can be found in Table 5.1 and Table 5.2. The requirements are organized based on the priority of the tasks, where tasks with a score over 80 is a must, tasks with a score over 60 is desirable but not critical and tasks with a score under 60 are features that are not required but can enhance the user experience.

The requirements specify a system that is intended to offer organizations flexibility in terms of user and project access. Users and administrators access the system using a personal email and password. This has the advantage of making it easy to both grant and remove access to the system and the email can be used for communication. A system user is linked to the organization of the users employee, and can manage projects and accounts linked to their organization. System administrators have full access to every project, user and organization in the system, and they are the only ones who can create new organizations. In addition to the described personal accounts for users and system administrators, every project also has an access token that can be used by anyone to access forecasts and project data using a REST API. This access token is created to make it easy for the users to integrate the service into their systems. The functionally related to user management is contained in requirements FR1-FR9, FR19-FR22.

The system should also offer a set of tools to validate forecasts and project data. The requirements specify that the system shall display general project information, live training progress, view a subset of project data and have graphs that can be used to validate data and forecast performance. This functionality is specified in requirement FR10-FR18, FR23-FR24 and is intended to make it easy for the user to create, view and validate the system forecasts, without any machine learning knowledge.

ID	Requirement	Priority (0-100)
FR1	A system administrator shall be able to create new organizations.	100
FR5	A user shall belong to one and only one organization.	100
FR2	A system administrator shall be able to invite users to join any organization.	95
FR9	A user shall be able to sign in using an email address and a password.	94
FR22	A user shall be able to get a reset password link sent to their email.	90
FR8	A user shall have a list of all projects that belongs to the users organization.	90
FR10	A user shall be able to create a new project, by uploading training data as a CSV file.	89
FR18	A user shall be able to see all information about a project in their organization.	88
FR11	A user shall be able to get forecasts up to 48 hours after the latest reported production.	85
FR16	A user shall be able to get forecasts using a REST API.	84
FR24	A user shall be able to upload production data using a REST API.	84
FR6	A user shall be able to invite more users to their organization.	81
FR19	A user shall be able to change password.	80
FR17	A user shall be able to get feedback while a new project is being trained.	75
FR12	A user shall be able to view the uploaded project data both as text and visualized in a chart.	70
FR24	A user shall be able to view the mean average error of the performed forecasts.	69
FR15	A user shall be able to view weather data for their project	68
FR14	A user shall be able to view recent system performance.	67
FR23	A user shall see a plot of future forecasts.	66
FR3	A system administrator shall be able to edit and delete users from the system	60

Table 5.1: List of functional requirements (1/2)

ID	Requirement	Priority (0-100)
FR4	A system administrator shall be able to edit and delete projects from the system	60
FR21	A user shall be able to remove users from their organization.	60
FR13	A user shall be able to upload their own forecasts and compare them to the forecasts of the system	59
FR7	A user shall be able to see all other users in the organization.	55
FR20	A user shall be able to remove projects from their organization.	39

Table 5.2: List of functional requirements (2/2)

5.1.2 Non-functional Requirements

To compete with existing solutions the system must be scalable, modifiable, reliable, performant, easy to use and cheap to operate. As the system is unlikely to get a lot of customers from the beginning, the system needs to adapt to different workloads. The system is also likely to experience highly variable traffic, as forecasts are typical made on a hourly basis and with new customers potentially requiring multiple projects to be set up simultaneously, model training can require immense resources. This means the system could see high traffic increases at specific times. In addition as the power producers rely on the forecasts to sell their power, it is important that the system is always available, and that down time is kept to a minimum.

The system is intended to be maintained by the AI department at TrønderEnergi. Since the AI department mostly consists of data scientists, the system must be made in such a way, that it requires very little maintenance and uses the tools, platforms and services they are familiar with. Therefore the system should use python as the main development language and automate the process of code delivery using a DevOps process.

If the system produces acceptable results, TrønderEnergi is also considering extending the system to other production facilities such as hydroelectric plants or solar plants. It is also possible that they want to change the training process to accommodate their own models. Therefore extra consideration must be taken in regard to modifiability of the system. Based on the properties described, a list of non-functional requirements has been developed and they can be found in Table 5.3. The requirements are intended to be validated, as such they are created to be as specific as possible.

ID	Requirement
NFR1	Modifications to the system shall be active in production within 15 minutes.
NFR2	An end user with a internet connection of 100Mbit/s, shall be able to get a new prediction within 5 seconds.
NFR3	An end user with a internet connection of 100Mbit/s, shall be able to load any page on the website within 2 seconds.
NFR4	It shall be easy for new users to use the system, a new user shall be able to setup a project and upload data within 5 minutes.
NFR5	A new project shall be trained and ready to accept forecast requests within one hour.
NFR6	The system shall automatically scale to accommodate 500 users, with one user joining every second.
NFR7	The system shall automatically scale to accommodate training of multiple projects, training of 20 projects started 10 seconds apart shall not take more time per project then training 10 projects also started 10 seconds apart.
NFR8	An experienced data scientist from the AI department at TrønderEnergi shall be able to make a modification to the wind power model training code without assistance from the original system developers.
NFR9	A software developer shall be able to extend the system to support other time series related problems, such as forecasting hydroelectric park production or grid loss, within 100 hours of development.
NFR10	A software developer shall be able to extend the system to support organizational permissions, such that only a select pool of users can manager the other users in the organization, this should be achieved in no more then 10 hours of development time.
NFR11	A cloud engineer shall be able to port the system to a different cloud provider within 50 hours.

Table 5.3: List of non-functional requirements

5.2 Deployment Platform

The system will be deployed as a set of microservices, therefore the deployment platform should be designed to deploy many smaller services, without extensive configuration and at low cost. Since managing physical infrastructure require a lot of expertise and work, we explored offers from cloud providers. The service selection was done by evaluating factors such as the preferences of TrønderEnergi, scalability, integration with DevOps services, ease of use, platform dependency, and cost.

TrønderEnergi was already using Microsoft Azure as their cloud provider and would like this system to do the same. Azure offer many servers that work well with microservices such as Azure Service Fabric, Azure Functions, Container Instances, App Service and Azure Kubernetes service (AKS). Table 5.4 list the Azure services that could be relevant for the project, based on the presented evaluation factors, excluding cost as it is hard to compare directly. Below each of the services will be presented and compared.

	Functions	Container Instance	App Services	AKS	Service Fabric
TrønderEnergi experience				X	
Scalability	X	X	X	X	X
Integration with DevOps	X		X	X	X
Ease of use	X	X	X		
Platform independent				X	
Community				X	

Table 5.4: Azure services compatibility

5.2.1 Azure Functions

Azure Functions is a serverless platform that can run event-triggered code without the need to manage infrastructure [36]. Functions is easy to get started with, it offer a good integration with Azure DevOps and has automatic scaling. As part of our specialization project we tested Functions using Docker to perform forecasts, Functions showed promising results, however it was not able to scale down to zero instances, meaning that it would always have a container running in the background accumulating cost. Scaling down to zero instances is however possible if the application does not use Docker, however that would also make the application less portable.

5.2.2 Azure Container Instances

Azure Container Instances can be used to run on-demand serverless Docker containers without the need to manage infrastructure [34]. Azure Container Instance is similar to Azure Functions, however it is designed to be an easy solution for deployment of containers and it does not offer the same event framework as functions does. In fact Container Instances can be used as a compute platform for both Functions and Azure Kubernetes Service(AKS). It also differs from virtual machines(VMs) as it does not require provision and it only accumulates cost as the container is running.

5.2.3 Azure App Services

Azure App services is designed as a fully managed platform for building and hosting web apps. App Services is easy to setup and requires little configuration [35]. It offers functionality such as backup, scaling, certificate management without the need for third party services. The disadvantage of App services is that its proprietary to Microsoft, so changing cloud provider is not an option, it is also designed specifically for web apps, as such it does not offer the same flexibility for containers as for instance Kubernetes or Service Fabric.

5.2.4 Azure Kubernetes Service (AKS)

Kubernetes is a container orchestration system created for automated deployment, scaling and managing containerized applications [29]. Kubernetes has emerged as an industry leader with over 66k stars on GitHub [19] and an active community. Many of the popular Cloud providers also offer managed Kubernetes implementations such as Amazon Web Service (AWS), Google cloud Platform (GCP) and Microsoft Azure. We would argue that it is harder to get started with Kubernetes then services such as App Service, Container Instances or Functions, however due to the large community and extensive documentation resolving issues can often be done with a Google search. Azure Kubernetes Service(AKS) which is the managed Kubernetes implementation by Azure, offers seamless integration with other Azure resources such as DevOps, networking and virtual machines. AKS is also competitive in regard to costs as Azure only charges for the use of virtual machines and not for the cluster management.

5.2.5 Azure Service Fabric

According to Microsoft [37]; Azure Service Fabric is a distributed system that makes it easy to package, deploy and manage scalable and reliable microservices. For our purpose Service Fabric can be compared to Kubernetes as they both offer much of the same functionalities and both have all the functionality needed for this project. However Kubernetes is a more popular platform and has a larger community then Service Fabric and since TrønderEnergi is also using Kubernetes for other projects, we think it will be easier to find solutions if we have issues with Kubernetes, than with Service Fabric. Service Fabric would also be much harder to port to a different cloud platform as neither AWS or GCS offers it as a service.

5.2.6 Comparison and choice of service

As part of our specialization project we created a prototype for the system using Azure Functions for forecasts, Azure App Service for the web interface and Container Instance for training. This allowed us to utilize the best of every platform, low cost for on demand training as well as easy setup and management of the web app and forecast functionality. However this also meant that editing the system would require knowledge of multiple tools and processes, and it made it difficult to configure additional containers such as PgBouncer and Redis, as the services does not have any direct connection. Form a cost perspective this solution also had an issue when the system was not in use, because Azure Functions and App Services would run all the time accumulating cost, making them more expensive then the price of one VM, that would have been the minimal requirement for Kubernetes. This difference could have been neglected if Function had been implemented without Docker.

To not overcomplicate the setup and select one platform that best fits the system, we decided to consider services that were better suited for lager microservice deployments. For this we considered the best choices to be either Service Fabric or Kubernetes, as they both are created to manage a lager clusters with many containers. However for this project we evaluated Kubernetes to better fit our needs, firstly because TrønderEnergi was already using it, second because of it large community and third because of its compatibility with other cloud providers. As a caveat Container Instance can be used to rapidly scale Kubernetes clusters, such that we can also utilize the best of serverless.

5.3 External Services

This section will present the external services that are used to power or enhance the functionality of the product. This includes services that are used to run the project such as Azure Kubernetes Service, but also services to get data such as Meteomatics.

5.3.1 Azure Kubernetes Service (AKS)

Azure Kubernetes Service (AKS) as explained above is a container orchestration system for containers, and will be used to deploy the system as a set of Kubernetes objects. Kubernetes object are persistent entities that are used to represent the state of the cluster. There are many objects for different purposes but a few examples that are relevant to the application is Deployments, Pods, Jobs and Cronjob.

A Kubernetes cluster can consist of one or more compute instances known as nodes, a node is a physical compute unit or a virtual machine that serves as a work unit in the cluster. For AKS the nodes are Azure Virtual Machines, and alternatively Azure Container Instances. Having more nodes can improve reliability, since the service can be deployed to multiple nodes at the same time, so if one node goes down the service does not. Nodes can also be added or removed from the cluster to scale to a given workload, this is known as horizontal scaling as presented in the background section.

5.3.2 Azure Database for PostgreSQL

Azure Database for PostgreSQL is a manged PostgreSQL service offered by Azure. The services comes with features such as automatic backup and scaling.

5.3.3 Azure File storage

Azure file storage is a storage services that enables applications to share the same storage space. For this project the file storage was used to offer a president volume share to different Kubernetes containers.

5.3.4 Azure DevOps

Azure DevOps consists of a set of services and tools that provides some of the fundamental functionality needed to work in a DevOps environment. This include code repositories, boards, wikis, code and release pipelines, artifacts and test plans.

5.3.5 Azure Container Registry (ACR)

Azure container Registry is a repository for Docker images similar to Docker Hub. ACR can store private images and can be connected to Azure DevOps and AKS. This means it can be used to store images built by Azure build pipeline, before it is deployed to AKS.

5.3.6 GitHub

GitHub was used to store all repositories related to the project. GitHub offers free private repositories that can be easily integrated with Azure. GitHub also offers testing and running of other scripts free of charge using GitHub actions.

5.3.7 Amazon Simple Email Service

Since Azure do not have a built in email service, it was decided to use Amazon Simple Email Service as it is competitively priced and we have previous experience with the service.

5.3.8 Meteomatics

Meteomatics is a provider of weather data that offer both historical data and future forecasts. This project uses Meteomatics to fetch weather data for all wind parks that are both used for training and forecasting purposes.

5.4 Implementation

This section will describe how the system was implemented using microservices and how it utilizes AKS and Azure DevOps. First a decision had to be made in regard to how such a system could be divided into a set of smaller services. As the system have three distinct operations; user and project management, model training and model predictions, it could be a natural way of dividing the system, that can offer a few advantages.

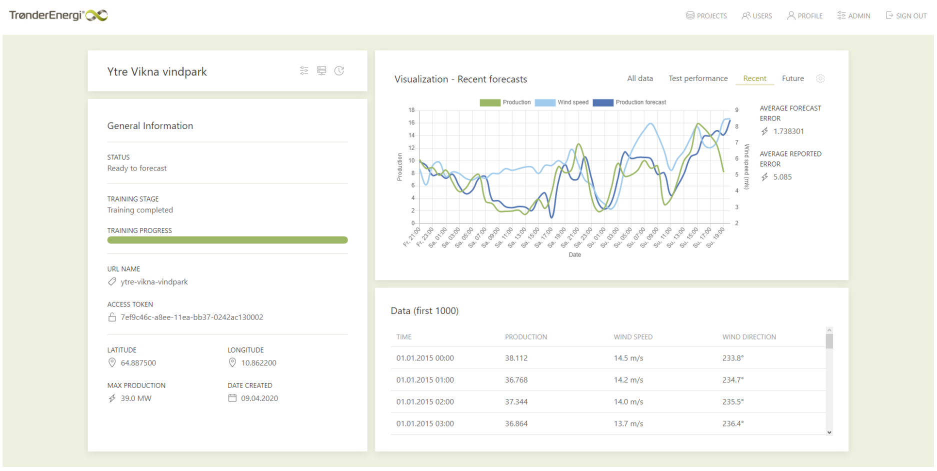
Firstly a data scientist can work on the training and prediction services without having to be limited by any design decision made by the developers creating the user and project management system. This allows them to use their own tools, repositories, deployment process, test framework, work processes etc.

Secondly theses services can be scaled individually such that that project training can be scaled without needing to scale the user and project Management Service. This is particularly useful for applications such as this, since it includes operations that require extensive resources for short amounts of time.

This section will first describe how each of the three services were implemented, and the additional microservices that are required to achieve the required functionality. Finally the section will present the tools and processes used for development and deployment of the system.

5.4.1 Management Service

The Management Service is intended to provide the external graphical user interface (GUI) of the system. The service is designed to simplify user and project management by providing users with an accessible web page that gives users the tools needed to create and manage users, create new projects, validate existing projects and compare results with their own forecasts. Figure 5.1 shows the project page from the Management Service to illustrate some of the project functionality it provides. Appendix A.1 contain a full list of screenshots from the Management Service to illustrate the full functionality of the system.



Ytre Vikna vindpark

General Information

STATUS
Ready to forecast

TRAINING STAGE
Training completed

TRAINING PROGRESS

URL NAME
ytre-vikna-vindpark

ACCESS TOKEN
7ef9c46c-a8ee-11ea-bb37-0242ac130002

LATITUDE	LONGITUDE
64.887500	10.862200

MAX PRODUCTION	DATE CREATED
39.0 MW	09.04.2020

AVERAGE FORECAST ERROR
1.738301

AVERAGE REPORTED ERROR
5.085

Data (first 1000)

TIME	PRODUCTION	WIND SPEED	WIND DIRECTION
01.01.2015 00:00	38.112	14.5 m/s	233.8°
01.01.2015 01:00	36.768	14.2 m/s	234.7°
01.01.2015 02:00	37.344	14.0 m/s	235.5°
01.01.2015 03:00	36.864	13.7 m/s	236.4°

Figure 5.1: Screenshot - Management Service - project page

As presented in the functional requirements, system administrators requires full access to all projects and users. Therefore the Management Service must also be able to customize the user experience based on the permissions of the authenticated user.

Even though one of the advantages of microservices is the ability to use multiple languages, the goal of this system is to use python as much as possible, such that it is easy for the AI department at TrønderEnergi to maintain the system. To achieve this goal, the application was developed using Django, a feature-rich python framework that comes with a set of useful tools and features that helps speed up the process of making dynamic web-pages.

One of these features are the included user and authentication functionality. It can be highly customized, includes password hashing and salting using the PBKDF2 algorithm with a SHA256 hash. To optimize the functionally for this system, only a few changes had to be made; firstly username had to be changed to email, secondly a relation would have to be made to the organization of the user, thirdly all the provided authentication web pages, had to be styled to fit the overall application style.

Security is always important when handling sensitive data, in this case wind farm production data and personal employee data. Django has implemented solutions to some of the most common security risks. Firstly Django's template language protects against common attacks such as Cross site scripting (XSS) and Cross site request forgery (CSRF). Secondly the built in object-relational-mapper (ORM) protects against SQL injections. Finally Django protects the system against clickjacking. While having a secure framework is not the entire solution it is a good start for creating a secure site.

The Django framework introduces a concept known as apps or applications, that is a python package that provide a set of features, that could be reused in other projects. For the Management Service we have utilized this concept to create one application for accounts, that handles any feature that is related to user and organizational management, and one app that is responsible for project management. Each of the application uses the Django framework to process requests and to create or update database rows. This means the Management Service can easily be modified or scaled in the future by adding or removing applications.

Part of the job of managing users and projects is to manage the related database tables. All database elements are defined as python code, and Django uses a object-relational-mapper (ORM) to map python code to SQL quires. This has the advantage of making table creation and queries simple to understand for any python developer. As part of this setup Django uses migrations to propagating changes made to the database schema. This is an essential feature for easily updating the database schema, and since the Management Service has the management responsibility it is the only service that will update any database schemas related to user or projects.

When a user want to create forecasts for a new wind park, they first have to create a project. This project creation is done using the Management Service and involve uploading park information and training data. However the park also need historical weather data, this data is fetched by the Management Service using a REST API provided by Meteomatics.

The Management Service is not just one runtime instance but a set of contains that communicates to enable the functionality described above. The primary containers needed for the service are tightly coupled and are deployed together. Scaling is done on a deployment bases and as such all the primary containers are scaled together. The Management Service consist of the following containers:

Django Web

This is the primary container in the Management Service, it is responsible for accepting and processing requests form the users. It can accept HTTP requests and WebSocket requests, however due to how Django works HTTP requests are processed synchronous. Therefore this container uses the Celery container to offload any operations that requires a lot of time to be completed.

Nginx Reverse Proxy

This container is running Nginx and is used as a web server and reverse proxy. It serves the user with static content such as images, CSS and JavaScript and routes all other requests to the Django web container.

Celery Asynchronous Worker

This container is used to perform asynchronous operations using Celery. It is for instance used for fetching weather data when new projects are created, this allows the Django web container to send a response to the user and let the process complete in the background.

Other Containers

When the Training Service is training a new model, it sends updates to the Management Service. The Management Service needs some functionality to receive these updates, this is done with a container that continuously watch's for updates. This container is responsible for listening to Redis (message broker explained in section 5.4.4) for updates from the Training Service, if such an update is received, it updates the database and uses a third-party package known as Django Channels to update any active WebSockets connections.

5.4.2 Training Service

The Training Service is responsible for creating machine learning models for new projects and to update existing models with the goal of increasing model accuracy. The Training Service is not running all the time, but can be invoked to train a single model. If multiple projects requires training at the same time, one job is created for each project. The model trained is a TL-GTB model utilizing the 10 wind farms with shortest geographical distance from the target wind farm as source wind farms using the ensemble source utilization strategy with equal weights. This model implementation is based on experiments presented in chapter 6 with results and justification chapter 7. An example of Training Service usage is included in Appendix A.2.

The primary purpose of the Training Service is to train new models, however it also has other responsibilities. As part of the training process the Training Service conduct a performance test. The performance test creates forecasts for a set of test data, which is held out from model training. These forecasts are saved to the shared database and are used by the Management Service to visualize the forecasting accuracy to the customer.

The Training Service is heavily data reliant, and it communicates with the postgreSQL database created by the Management Service. Since multiple services needs access to the same database, we created a private python library known as Forecasting as a Service(FaaS) database manager that simplifies database operations by adding a layer of abstraction. This means that in most cases when there are changes to the database, only the library needs update and not each individual service.

When a new model is being trained, information about the training progress is sent to the Management Service. The information includes how much of the training is completed and a message to let the user know how far training has come. This information is sent using the publish-subscribe pattern with Redis as the message broker. This solution was selected to decrease the number of dependencies and the number of point-to-point communications in the system, as this would help reduce technical debt according to de Toledo et al. [11].

5.4.3 Inference Service

The primary functionality of the Inference Service is a REST API that is designed to be accessed by other systems. The main purpose of the API is to collect production data and to make production forecasts, but it also offer other information. To create new forecasts the Inference Service uses the model trained by the Training Service. The service is accessed using the project URL and a unique access token for each project, that can be acquired from the Management Service. An example of Inference Service usage is included in Appendix A.3.

For the Inference Service to be able to make forecasts it needs production data. This data will be uploaded to the Inference Service using its REST API. To ensure quality forecasts the customer should upload production data as often as possible, ideally once an hour. This would typically be implemented in the system of the customer as a scheduled job.

When the Inference Service is used to perform system forecasts, the user can choose between getting a single forecasts or multiple forecasts. A user may request a forecast for a single date by setting the date query string, as long as the date is within 48 hours after the latest reported forecast. If the user does not specify a date, the user will be given forecasts for the full 48 hours window.

To speed up the development process of the Inference Service a web framework had to be selected. Unlike the Management Service, this framework should not have features such as administration centers or advanced authentication systems. The framework should be primarily made for REST APIs, be easy to use, and should be easy to customize. A framework that fit the requirements well was FastAPI. FastAPI was selected for the Inference Service as it offer just what is needed, and not much more.

The secondary objective of the Inference Service is to perform forecasts that will be used by the Management Service to visualize model performance. These forecasts are scheduled every hour and forecasts are performed for the next 48 hours from the last reported production. This is to ensure that the Management Service still receive forecasts, even if the park did not report production for a few hours.

Just like the Training Service, the Inference Service also communicates with the shared database using our private database manager library. The database access is used for features such as validating the access token, fetching project data and updating the project with the newest predictions.

5.4.4 Other Services/System utilities

In addition to the three main services the system also consists of many smaller units that are essential to deliver the required functionality. Each of these smaller services will be presented below with their relation the the main services.

PostgreSQL Database

The PostgreSQL database acts as a shared data storage for all the services. The database stores data about users, organizations and projects, including production data. The database is managed by Azure to minimize configuration, and take advantage of the included features such as backup.

Shared File Storage

The system uses Azure File Storage to store the models trained by the Training Service. Both the Inference Service and the Training Service have direct access to the storage, such that new models are available to the Inference Service instantly.

PgBouncer

PgBouncer is a connection pool that is used to connect one or more clients to the same database using only one connection. PgBouncer was added for two reasons. Firstly because Azure Database for PostgreSQL has a long and highly unpredictable connection time; one second is not uncommon, making the request time problematic for services that are directly used by the end user. Secondly PgBouncer was added to ensure performance as the application scale.

Redis

Redis is an in memory key-value database that can be used as a cache and message broker. For this project it is used as a message broker for the Training Service and the Management Service. In addition it is also used internally in Management Service as part of Django-channels to enable asynchronously websockets in a distributed system.

Nginx ingress and Cert Manager

In Kubernetes an ingress is a Kubernetes object that can manage HTTP and HTTPS connections to services inside a Kubernetes cluster. An ingress requires an ingress controller, for this project we selected Nginx due to its large community and ease of use. The ingress was then configure to direct requests to the Management Service and the Inference Service. To enable HTTPS the domains for each service also requires a valid transport layer security (TLS) certificate, this was accomplished using Cert Manager that automatically configures TLS certificates using Let's Encrypt.

Weather Fetching

In addition to the historical weather data that is required for training, all projects also requires updated forecasts on a daily basic. Therefore we have created a Cronjob that fetches weather forecasts from Meteomatics once a day.

5.4.5 Developing and Deploying the System

This section will first explain how the system utilizes Docker and Kubernetes to develop and deploy the system. Secondly this section will discuss how the system take advantage of GitHub and Azure DevOps to automate the process of deploying changes to AKS.

Local Development

Kubernetes can offer a lot of useful features in a production environment, however its extensive feature set can also make local development difficult. The system will leverage Docker due to it popularity and compatibility with Kubernetes and other cloud products. For local development the system will use docker-compose, a tool created by Docker to setup and run multi-container Docker applications. This allows developers to easily run the application locally without the overhead of using Kubernetes. Each service will have its own docker-compose file, with all the containers needed for that service, also including applications such as Redis or PostgreSQL. This configuration allows each service to be developed in isolation.

Deployment to Kubernetes

As presented in subsection 5.3.1 resources are deployed to Kubernetes in the form of different objects. The web services for both the Management Service and the Inference Service are deployed as Kubernetes Deployments. A Kubernetes Deployment uses a ReplicaSet to scale one or more pods. A pod is a Kubernetes object that can consist of one or more containers that will run on the same machine, at the same time. Pods are therefore ideal for containers that have a high level of cohesion, such that they can be scaled together. For instance the primary containers of the Management Service are deployed as one pod. In addition to the Deployment object, the presented services are also using an object known as Service, it allows the deployments to be accessible from other Kubernetes services.

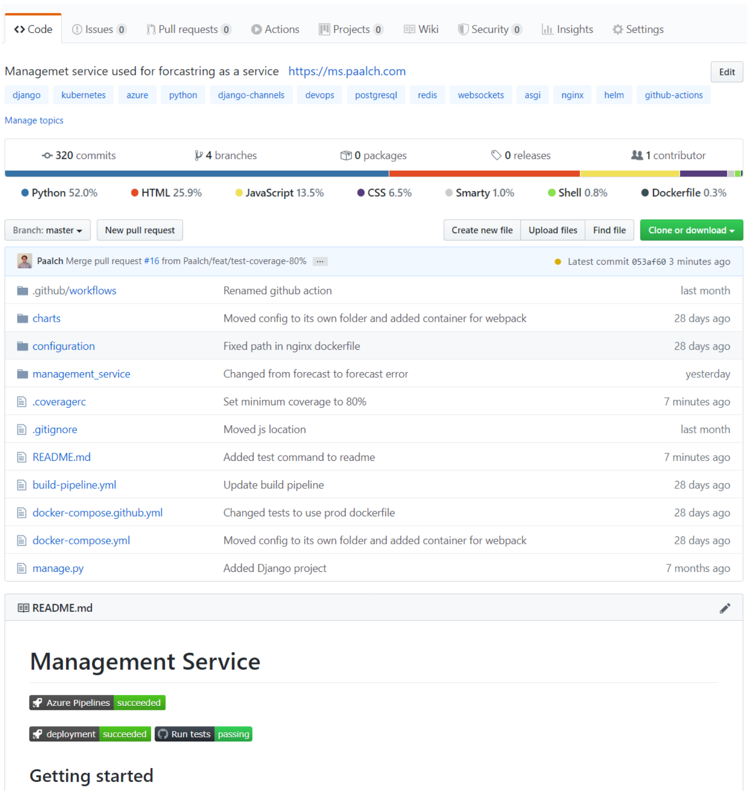
To automatically scale the Management Service and the Inference Service the system utilizes the Kubernetes object known as HorizontalPodAutoscaler. It tracks the average CPU utilization of each of the services, and scales the number of replicas up or down based on the desired CPU usage for each container.

Kubernetes also comes with a Secret object that is used by the system to store credentials, such as database credentials, Meteomatics credentials and the TLS certificates. Theses secrets can be injected into any container that requires them.

For tasks that are intended to run until they are completed, Kubernetes offer the Task object, it is used by the Training Service. The processes that are deployed periodical on a schedule uses the Cronjob object, they can run one or more Tasks given a schedule.

Code Repositories and CI/CD

All code created for the project is hosted on GitHub. To make the three main services easy to maintain individually, each service has it own private repository. In addition we have created a repository for the FaaS database manager and one for the configuration of the other smaller services. Figure 5.2 shows the GitHub repository for the Management Service, it includes a link to the service, a list of topics, a quick setup guide and a set of badges to see the current status of build, deployment and testing.



Managemet service used for forcastring as a service https://ms.paalch.com

320 commits · 4 branches · 0 packages · 0 releases · 1 contributor

Python 52.0% · HTML 25.9% · JavaScript 13.5% · CSS 6.5% · Smarty 1.0% · Shell 0.8% · Dockerfile 0.3%

Paalch Merge pull request #16 from Paalch/feat/test-coverage-80% … Latest commit 053af60 3 minutes ago

.github/workflows	Renamed github action	last month
charts	Moved config to its own folder and added container for webpack	28 days ago
configuration	Fixed path in nginx dockerfile	28 days ago
management_service	Changed from forecast to forecast error	yesterday
.coveragerc	Set minimum coverage to 80%	7 minutes ago
.gitignore	Moved js location	last month
README.md	Added test command to readme	7 minutes ago
build-pipeline.yml	Update build pipeline	28 days ago
docker-compose.github.yml	Changed tests to use prod dockerfile	28 days ago
docker-compose.yml	Moved config to its own folder and added container for webpack	28 days ago
manage.py	Added Django project	7 months ago

README.md

Management Service

Getting started

Figure 5.2: Github repository for Management Service

Before any code can be merged into the master branch of the Management Service, the automated tests must pass, with a coverage of at least 80%. This is done using GitHub Actions, currently it is only used by the Management Service, but it can also be added to the other service as they become more complex and manual testing is no longer sufficient.

After the code of any repository is merged to the master branch, it will automatically trigger the Azure build pipeline. It is responsible for building the Docker images required for the repository and/or package the provided Helm charts. Helm charts are responsible for configure how the services will be deployed to Kubernetes. In the case of the FaaS database manager the build pipeline is responsible for publishing the code to Azure Artifacts, so that it can be installed by other services using pip. After the build is finished it will trigger the release pipeline that will deploy the service to Kubernetes.

5.5 Architecture

The system architecture will be presented using the 4 + 1 architecture view model created by Kruchten [26]. The 4 + 1 architecture view model uses multiple concurrent views, to separately address the needs of various stakeholders such as the end user, developers, system engineers and project managers. Figure 5.3 illustrates the relation between the views.

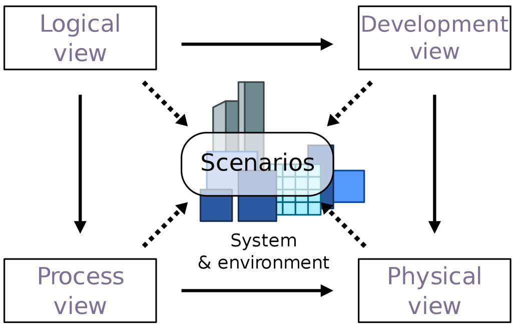


Figure 5.3: Illustration of 4 + 1 architecture view model from Dekker [12]

5.5.1 Logical View

The logical view is primarily designed to present the system in form of its functional requirements. The view is intended to support the interests of the end-user. For this system it is achieved by a set of views that illustrates the system functionality based on the system services presented above. A common way of presenting the logical view is by using UML class diagrams, however in this case it is not sufficient for covering the functionality of the system as multiple parts of the system are created as python functions and not classes. The logical view will therefore be presented with a custom diagram that present an overview of the system functionality.

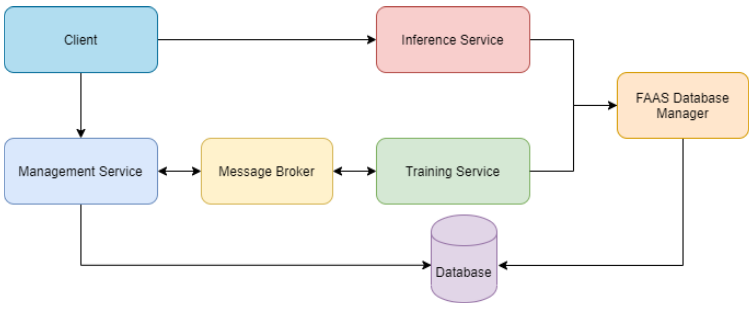


Figure 5.4: Logical view - system

Figure 5.4 illustrates the three primary services of the system, and how they communicate. As presented in section 5.4, both the Inference Service and the Management Service can be accessed by the end-user over the internet. The Training Service can only be accessed by the Management Service using Redis as a message broker. All services got the same access to the database, but while the Training Service and Inference Service is using a shared library (FaaS database manager), the Management Service got direct access using the Django ORM.

Management Service

Figure 5.5 illustrates the Management Service in terms of its primary functionality, it is divided into two applications; Projects and Accounts. Each of the applications was created using the same architecture patterns, but with the goal of implementing a specific feature set. The accounts application is responsible for all features that are related to user and company management, while the project application is responsible for all aspects of project management.

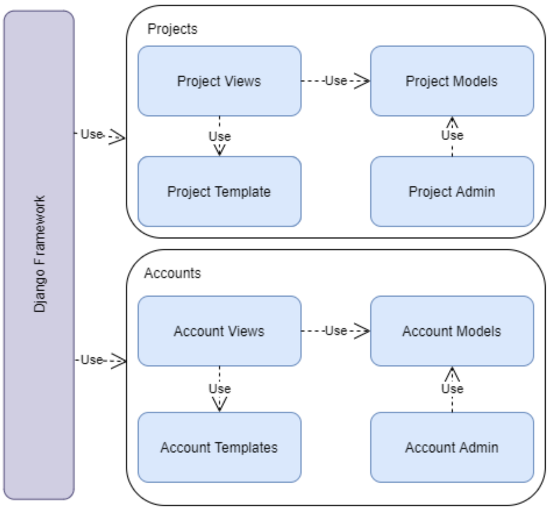


Figure 5.5: Logical view - Management Service

Figure 5.5 presents four concepts for each app; views, templates, models and admin. These concepts represent the standard workflow for a web application in Django. The view and template are used to process and return a response in form of a web page for a given request, the model is responsible for business logic and storage, and the admin represents the customization that can be done to the automatically generated admin user interface. This workflow is an architecture pattern known as model-view-controller (MVC), however due to the names in the Django specific implementation, it is commonly refereed to as model-template-view (MTV)[13]. This is because the model part of MTV is equivalent to the model in MVC, however the combination of the view and template in MTV are closely related to the view in MVC, as they are responsible for the presentation of data. The controller of the MVC pattern can be compared to the framework itself, as it is responsible for requesting the appropriate view.

Training Service

The Training Service is responsible for training new models, validate model performance, save the model to a shared storage, inform the Management Service of training progress and to update the database with test results. Figure 5.6 illustrates how the Training Service is organized in order to have these features. The training module is responsible for training and evaluating the model, it uses the provided utilities to notify the Management Service on training progress and to save the trained model, in addition it uses the FaaS database manager to get project data and to save tests results.

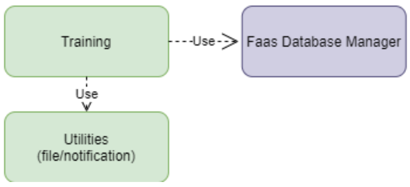


Figure 5.6: Logical view - Training Service

Inference Service

Much like the Management Service, the Inference Service design is also influenced by the underlying web framework. The Inference Service utilizes FastAPI to create a REST API that use JavaScript Object Notation(JSON). Figure 5.7 shows the primary elements of the Inference Service, responsible for processing requests form the user and perform forecasts. The figure includes a module called ”Project Router”, it contains all the functions used to process requests from the user. It uses the Inference module, for requests that is related to forecasts. The figure illustrates that both the Inference module and the ”Project Router” uses the FaaS database manager. The Inference module uses the database manager to get project data, while the ”Project Router” uses the database manager to validate access tokens, get project meta data, and to get project data, all depending on the user request. Finally the ”Request/Response Types” module includes a set of classes, that specifies the format for requests and response to the API itself.

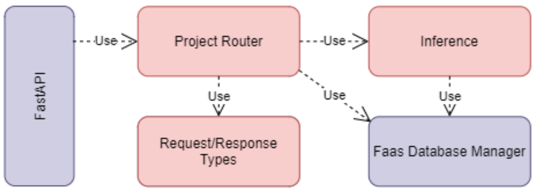


Figure 5.7: Logical view - Inference Service

Entity-relationship model (ER)

The goal of the system is to provide energy producers with accurate production forecasts using machine learning. This is an application that is very reliant on data, and therefore we have also included a entity-relationship (ER) model to show how the system stores data. The model can be seen in Figure 5.8. The ER diagram shows the relationship between the entities of the system. It illustrates how a company can have many users and projects, and that a project can have many project data instance. A single project data instance stores data for a single date, and as the forecasts are improved with larger data sets, each project should ideally have as many project data instances as possible.

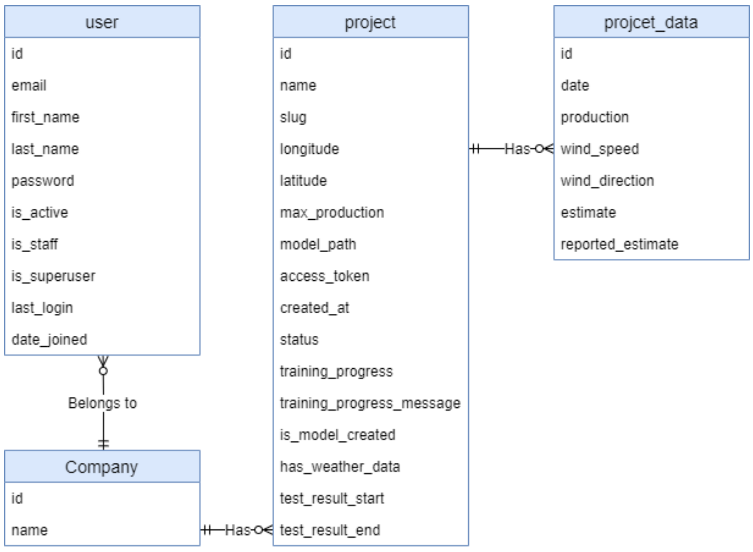


Figure 5.8: Entity-relationship model (ER)

5.5.2 Development View

The development view is concerned with the software organization of the system, and is intended to support software developers. This view will be presented using a UML package diagram for each service, that will also include the package artifacts and primary script files. The concepts presented in the logical view should be relatable to the primary packages.

Management Service

The package diagram for the Management Service can be seen in Figure 5.9, due to the size of the diagram, the figures for accounts, projects and JavaScript(js) are created as separate figures. Accounts can be seen in figure Figure 5.10, projects in Figure 5.11 and js (JavaScript) can be seen in Figure 5.12.

Figure 5.9 covers the overall structure of the Management Service and is highly influenced by the Django framework. The root package contains an app package with the functionally explained in the logical view, a set of configuration files, a package with assets that contains the JavaScript, a static folder that contains files to be served directly by the Nginx webserver, a set of root templates used by the apps, a test folder for tests, a test_data folder that is used to hold data used for development and a util folder used for utilities.

The files contained in the root package are all concerned with configuration of the framework. The ASGI (Asynchronous Server Gateway Interface) file is used to communicate with the web server forwarding requests to the Django framework. By default Django uses WSGI (Web Server Gateway Interface), however as the project adapted WebSockets it required an interface with support for asynchronous operations and the system changed from WSGI to ASGI.

The url.py file in the root package is used for routing, it is responsible for routing all requests to its appropriate view, module or app. In Django it is common to have multiple url.py files, one root and one for each app. Therefore both accounts and projects have their own urls.py. The routing.py file has a similar purpose and is used for routing WebSocket connections. Finally the settings file is used to configure the application, this includes aspects such as configuring the database, Redis, environment, templates, apps, third-party modules, middleware, allowed hosts etc.

Figure 5.9 also contains a util folder that holds utilities that can be used by multiple apps. Currently it contains a utility for communicating with Meteomatics and the training_manager file that uses a manager from the training_manager package to start a new training job. The training manager was designed this way to support more then one platform.

In addition to the accounts and projects apps that will be covered below, the Management Service also has a third app, celery worker. It only contains the configuration used by the Celery, and is created as an independent app to follow best practices.

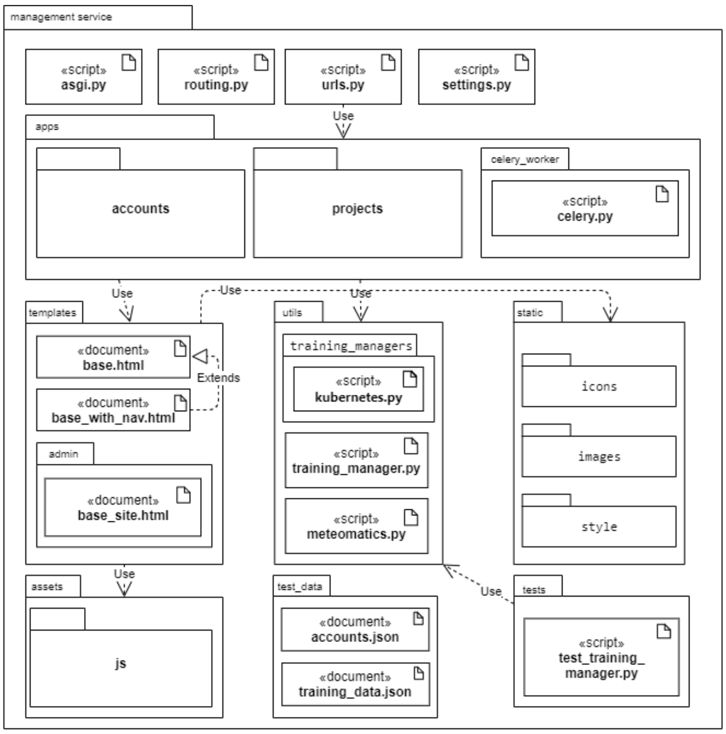


Figure 5.9: Development view - Management Service

Figure 5.10 shows the package diagram for the accounts app, the package provides functionality that is concerned with user and company management. The logical view introduced concepts such as models, views and templates, they can be seen as packages in the diagram. The templates are based on inheritance and most of the templates extends the base_with_nav.html or base.html template from the root template package. The templates in the registration package are used to overwrite the default Django functionality and are concerned with authentication process. The diagram also includes a package for tests, and one for migrations. The migration package holds migrations files generated by Django, the migration files are responsible for updating the database schema when a change is made to a model.

The accounts app also contains a set of files that are common in most Django applications. The admin file is used to configure the built in administration panel, it is used to specify what data is included and how it is displayed. The apps file is used to register the package as a Django app (part of the framework configuration), the url.py files extends the url.py file in the root package, the form.py is used to validate user input and the activate_user_email.py file is specific to the accounts application and it sends a welcome email to a given user.

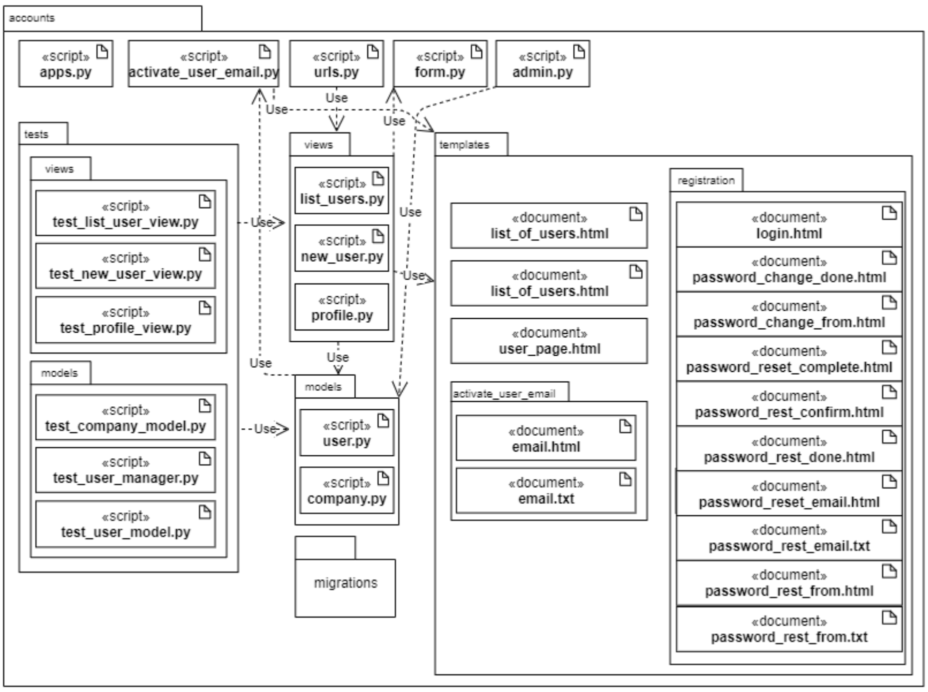

accounts
«script» apps.py
«script» activate_user_email.py
«script» urls.py
«script» form.py
«script» admin.py
Use
tests
views
«script» test_list_user_view.py
«script» test_new_user_view.py
«script» test_profile_view.py
models
«script» test_company_model.py
«script» test_user_manager.py
«script» test_user_model.py
views
«script» list_users.py
«script» new_user.py
«script» profile.py
models
«script» user.py
«script» company.py
migrations
templates
«document» list_of_users.html
«document» list_of_users.html
«document» user_page.html
activate_user_email
«document» email.html
«document» email.txt
registration
«document» login.html
«document» password_change_done.html
«document» password_change_from.html
«document» password_reset_complete.html
«document» password_rest_confirm.html
«document» password_rest_done.html
«document» password_reset_email.html
«document» password_rest_email.txt
«document» password_rest_from.html
«document» password_rest_from.txt

Figure 5.10: Development view - Management Service - accounts application

Figure 5.11 shows the package diagram for the project app, the package provides functionality that is concerned with project management. The organization of the package and files are similar to accounts, however the project app also contains a few other features.

The package contains a decorator.py file that is used to create custom decorators to be used by view functions. A decorator is used to alter the functionality of a function, without changing the source code of the function. One common usage for decorators in Django is to modify the access to resources, in this case this could for instance be to only allow employees from a given company to access a project.

The package also contains a task.py file, which is used to define functions that should be ran by Celery. So far it only contains one file that uses the Meteomatics util to fetch weather data.

The channels package contains the code that is used for WebSockets. The routing.py package specifies the paths, just like urls.py for HTTP requests. The consumers process the requests and can be compared to the view functions for HTTP requests.

In addition to the default model behavior, the project app also includes a manager package, that is used to create custom Managers for Models. In Django a manager is the interface that provides database query operations. Because the Django ORM does not provide functionality for all possible SQL quires, a new manager can extend the functionality of the base model manager. In this case this was done to implement a custom bulk update or insert methods for project data.

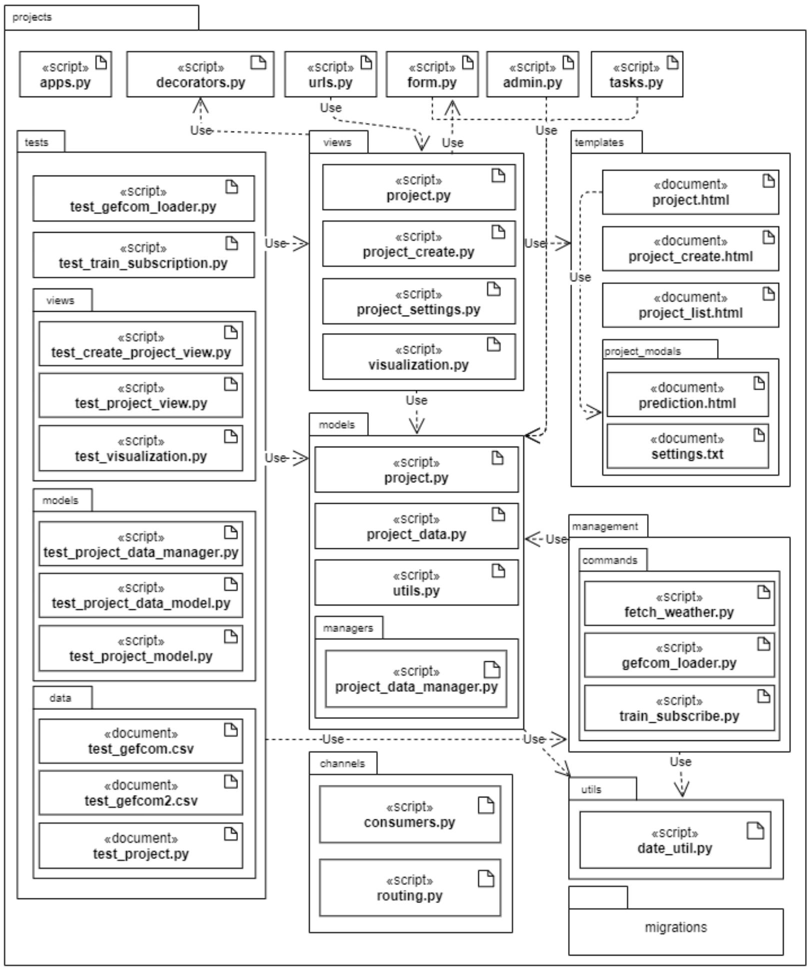

projects
«script» apps.py
«script» decorators.py
«script» urls.py
«script» form.py
«script» admin.py
«script» tasks.py
Use
tests
«script» test_gefcom_loader.py
«script» test_train_subscription.py
views
«script» test_create_project_view.py
«script» test_project_view.py
«script» test_visualization.py
models
«script» test_project_data_manager.py
«script» test_project_data_model.py
«script» test_project_model.py
data
«document» test_gefcom.csv
«document» test_gefcom2.csv
«document» test_project.py
views
«script» project.py
«script» project_create.py
«script» project_settings.py
«script» visualization.py
models
«script» project.py
«script» project_data.py
«script» utils.py
managers
«script» project_data_manager.py
channels
«script» consumers.py
«script» routing.py
templates
«document» project.html
«document» project_create.html
«document» project_list.html
project_modals
«document» prediction.html
«document» settings.txt
management
commands
«script» fetch_weather.py
«script» gefcom_loader.py
«script» train_subscribe.py
utils
«script» date_util.py
migrations

Figure 5.11: Development view - Management Service - project application

The project also features several dynamic web elements, such as buttons, modals and graphs; to power this functionality the project uses JavaScript. The JavaScript files are directly related to the templates containing HTML and are imported using the script tag. Figure 5.12 show the structure of the JavaScript files, dependencies and packages. The majority of the JavaScript is used on the project page, the project page uses all the files in the project package as well as index.js and navbar.js. It would be inefficient to load all those nine files individually, therefore the project uses Webpack to bundle the JavaScript before it is served to the user. The bundling is triggered as part of the deployment of the Management Service, and the finished bundles are served by Nginx web server. In development Webpack run in its own container, continuously watching for changes, such that a change to a file will automatically trigger a rebundle.

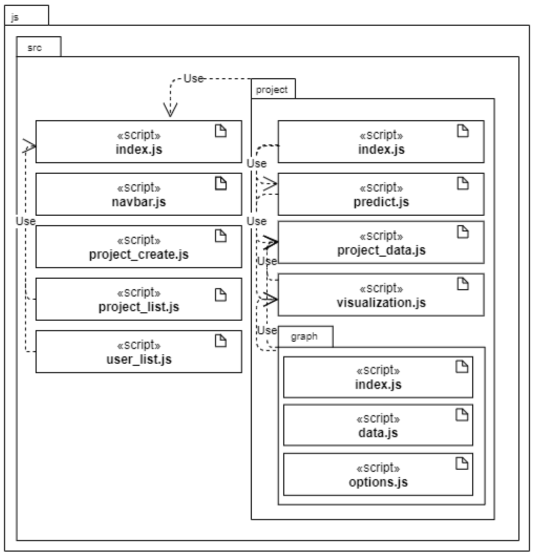


Figure 5.12: Development view - Management Service - JavaScript

Training Service

Figure 5.13 shows the package diagram for the Training Service. The two files in the root package are responsible for configuration the service. The training package is responsible for training, it consists of a training.py module, responsible for the model training pipeline, using the tl_catboost.py module, which implements the TL-GTB method, and feature_map.json, which defines the features used by the model. The training packages use the utility package to handle generic operations such as notifying the Management Service and saving files/models.

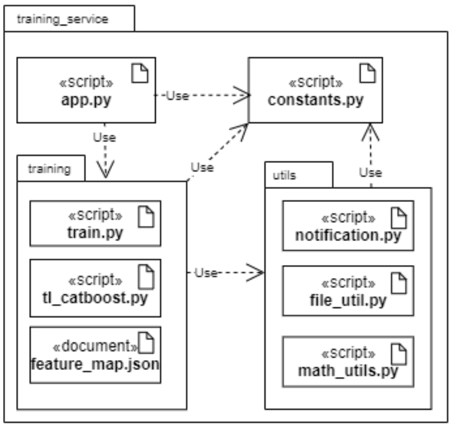


Figure 5.13: Development view - Training Service

Inference Service

Figure 5.14 shows the package diagram for the Inference Service. Similar to the Training Service it has two files in the root package that are used for configuring the service. The constant file includes global variables such as logging level and the path to the folder holding the project models. The main app is used to initiate FastApi and configure it to the specifications of the project. The utils package contains one file that is used to make forecasts for all projects, it is used for the Cronjob that performs forecasts for all project every hour. The inference package is responsible for performing forecasts. The project.py file in the router package is responsible for processing requests form the user. Finally the project.py in the request types are used to specify the format of requests and responses from the API.

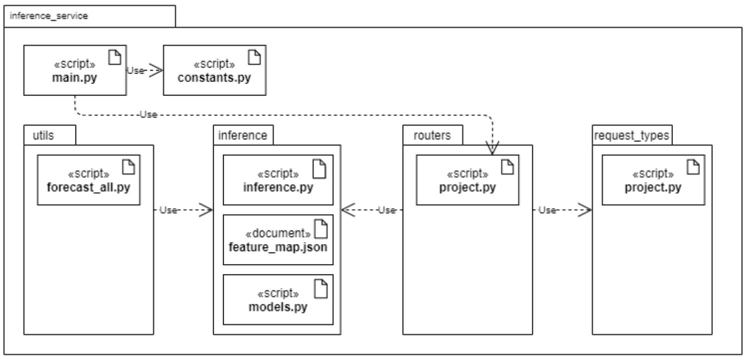


Figure 5.14: Development view - Inference Service

Forecasting as a Service Database Manager

The FaaS database manager is divided into three parts; the connector.py that is responsible for connecting to the database, the files in the models package that are responsible for building quires and the manage.py that is responsible for binding them together. All the models inherit from the abstract_model.py as it contains a set of generic SQL methods such as get and update. The files that uses the abstract_model such as projects.py contains the table name and fields of the project table and any custom methods used for that table.

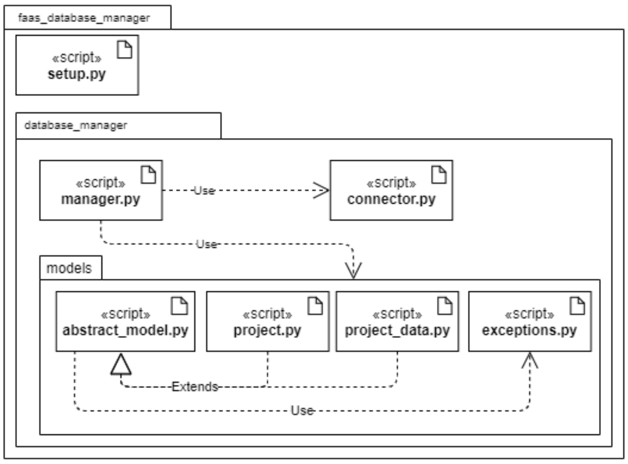


Figure 5.15: Development view - FaaS database manager

5.5.3 Process View

The process view is concerned with the dynamic parts of the system, how services communicates and the runtime behavior of the system. The goal of the process view is take into account some of the non-functional requirements. This section will present the request flow for the web services as well as a sequence diagram covering the process of training a new project.

Figure 5.16 illustrates how the Management Service processes a request from a web browser. First the request is sent to the Nginx instance acting as a web server and a reverse proxy. If the browser requests a file such as images, CSS or JavaScript the asset is returned directly by the Nginx instance. However, if the request is for any other resource, the request is forwarded to the Django application. Daphane is an Asynchronous Server Gateway Interface (ASGI) server and is responsible for running the Django application. The request is then processed by Django Channels, that determines the type of the request (HTTP/WebSocket). If the request is using WebSockets it is routed to a WebSocketConsumer based on the request URL. The WebSocketConsumer then communicates with the client using strings that are interpreted as JSON.

The request can also be using HTTP, if so the request will be processed using the normal Django request flow. The request is first sent to the request middleware. In Django, middleware is essentially a set of functions that are hooked into the request/response flow. The figure shows the basic hooks, however other hooks also exists. A majority of the default middleware is concerned with security, but it also does other things such as adding the user object to the request.

After the request is processed by the middleware it is routed to the appropriate view function based on the URL. The view is responsible for processing the request made by the user, it uses data from the models, possibly some utilities, and renders the content using a template. The response is finally processed by the response middleware and sent back to the client.

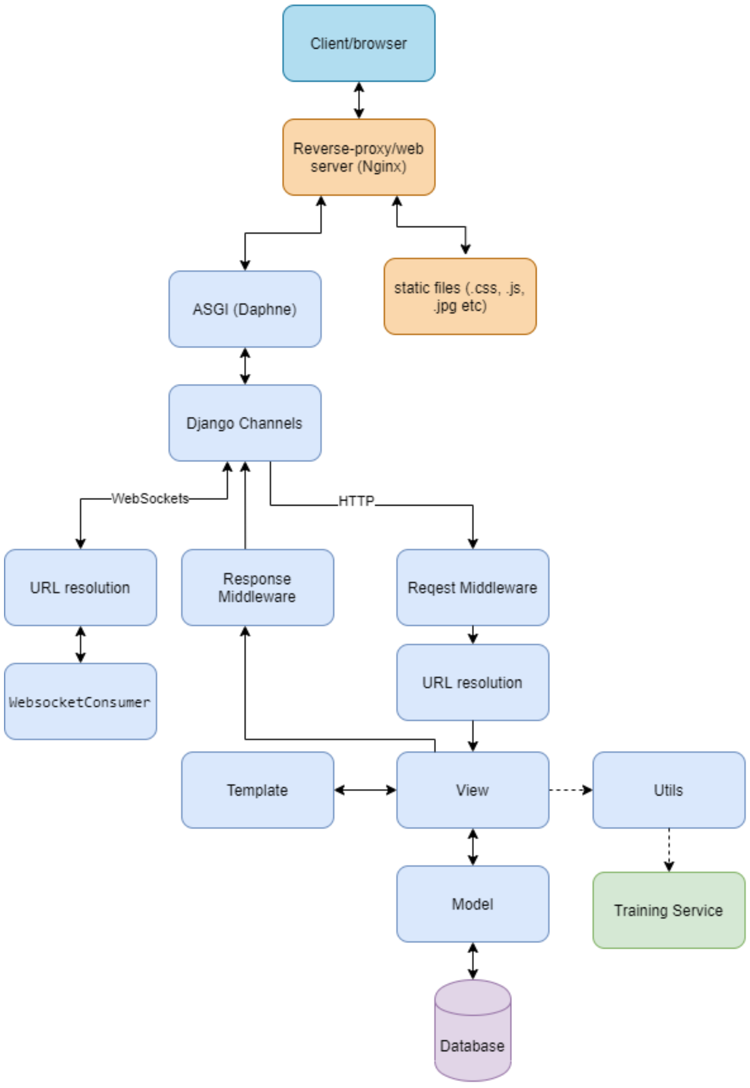


Figure 5.16: Process view - Management Service

Figure 5.17 illustrates a similar request to the Inference Service. As the Inference Service is not using static assets, HTML or authentication, the request flow is simpler than for the Management Service. For the Inference Service the request is directly forwarded to the Uvicorn ASGI server. The Management Service uses different ASGI servers, as the frameworks had different recommendations. However it should be possible to use the same WSGU server for both. Just like the Management Service, the Inference Service also has middleware, in this case it is only used for cross-origin resource sharing (CORS). CORS is used to allow browsers to access the API from other domains, in this case the domain of the Management Service. The request is then forwarded to a specific router function based on the URL. The function will process the request and use the FaaS database manager to acquire or update any data.

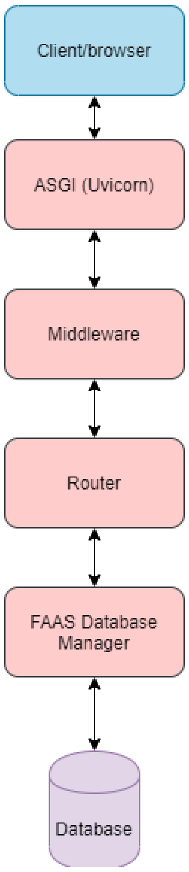


Figure 5.17: Process view - Inference Service

Figure 5.18 is a sequence diagram that outline the internal process of creating a new project. The figure illustrates the four containers that contribute to the creation of a new project. The creation starts as soon as the Management Service Web container have received the necessarily data from the client. The Management Service Web container will then create a new project and save the project data, before sending a request to the Celery container to start fetching the required weather data. When the weather data is acquired, the Celery container will initiate a new training job. The training will start; and as it progresses, it will send regular updates to the Management Service. Theses updates are received by the Train Subscription, that will forward them to the Web container, that is responsible for communication with the client. It is important to note that the system can have multiple instances of the Management Service Web and Celery (They are in the same pod), and that a new Training Service job is created for each project that is trained. However there are only one Train Subscription, that will map the messages received to the communication layer provided by Django channels/Redux.

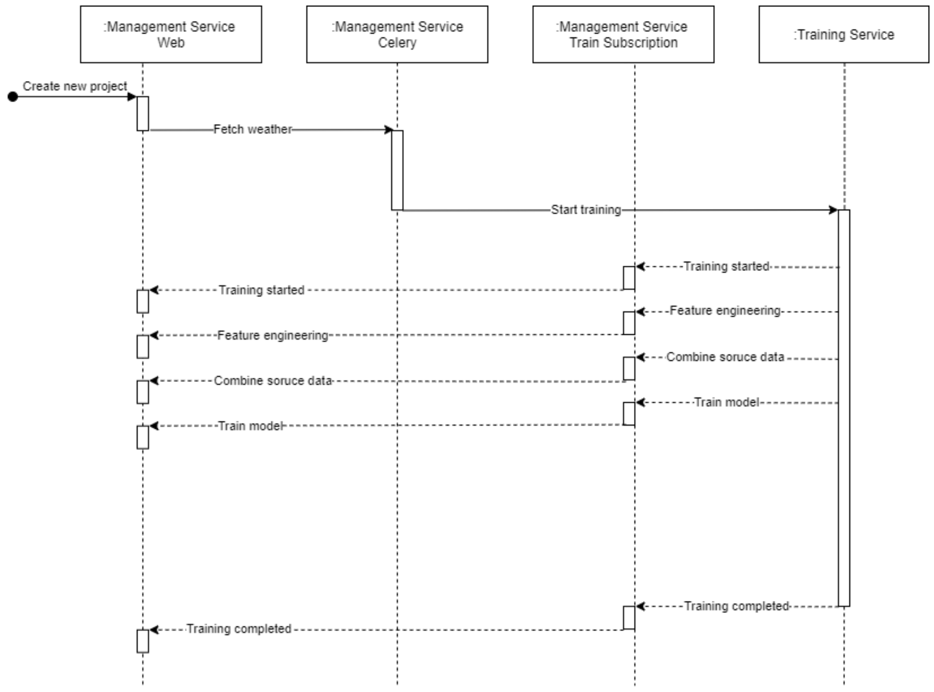


Figure 5.18: Process view - create project

5.5.4 Physical View

The physical view is concerned with the physical hardware and its mapping to software components. The physical view is intended to show the system from a systems engineer point of view. As this system is reliant on third party services, the physical view will be presented as the software components and their mappings to third party services, in particular Azure services.

Figure 5.19 illustrates the services used to build and deploy code and how they relate. Firstly changes made to the code base are pushed/merged to the master branch on GitHub. The event triggers Azure to build the required images, and pushes them to Azure Container Registry. The Build pipeline then triggers the release pipeline that will install the provided helm charts using Tiller, Tiller is the Helm server running in the Kubernetes cluster that is responsible for installing new charts. This is done by downloading the images created in the build step from Azure Container Registry and public images from Docker Hub (Nginx, Redis etc). Kubernetes will then start the new resources before finally terminating the old versions.

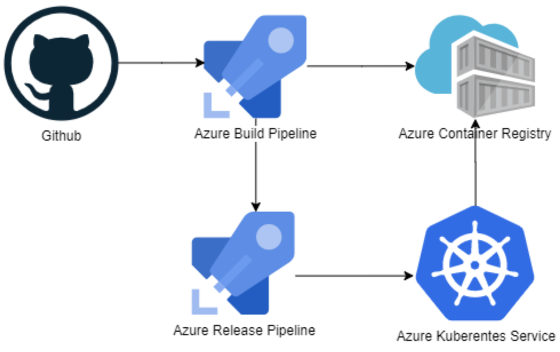


Figure 5.19: Physical view - DevOps

Figure 5.20 visualizes the infrastructure that is needed for the runtime operations of the system. The figure shows how the system is accessible from the internet using the Management Service and the Inference Service. When a request is sent to the cluster it is sent to the Nginx ingress, that is exposed to the internet through the Azure Load Balancer. The ingress will then route the request to the appropriate service, based on the domain name. The ingress is also responsible for encrypting the connection to the client using HTTPS, this is done by using a TLS certifcate that is acquired by Cert Manager, this process was described in section 5.4.4. The request can then be processed by the django-web-service in the Management Service or the only container in Inference Service. The services communicates using the shared PosgresSQL database, Redis or the shared file storage as can be seen in the figure. All connections to the database are done using the Pgbouncer services to reduce latency.

In addition to the resources that are directly responsible for processing user requests. The cluster also contains the Training Service that is initialized by the Management Service, and can run one or more jobs at any time. As well as cronjobs running at a set interval.

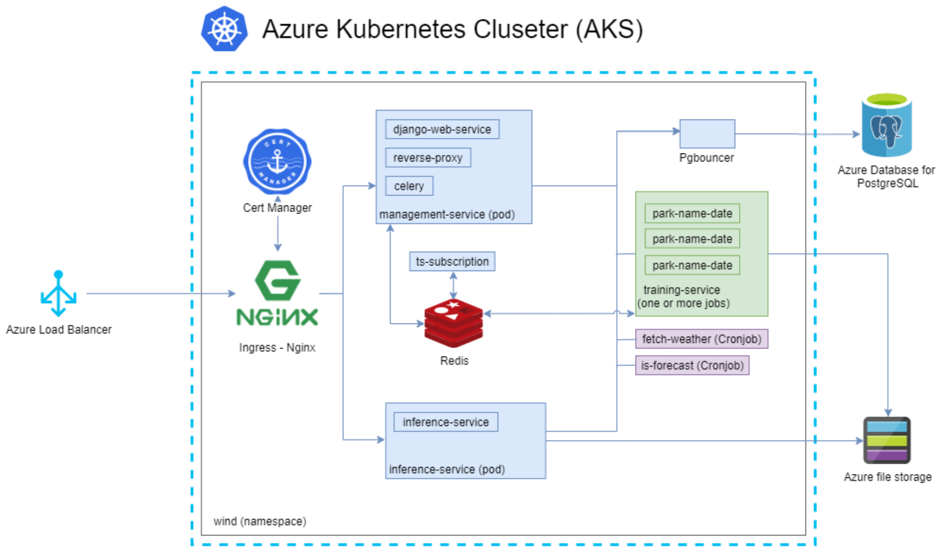


Figure 5.20: Physical view - runtime

5.5.5 Scenarios

The scenarios are the plus one of the model, they are intended to show a more high level abstraction of the functional requirements. We have implemented the scenarios in form of a user case diagram. Figure 5.21 show the two actors of the system; the customer and the administrator and what features they require.

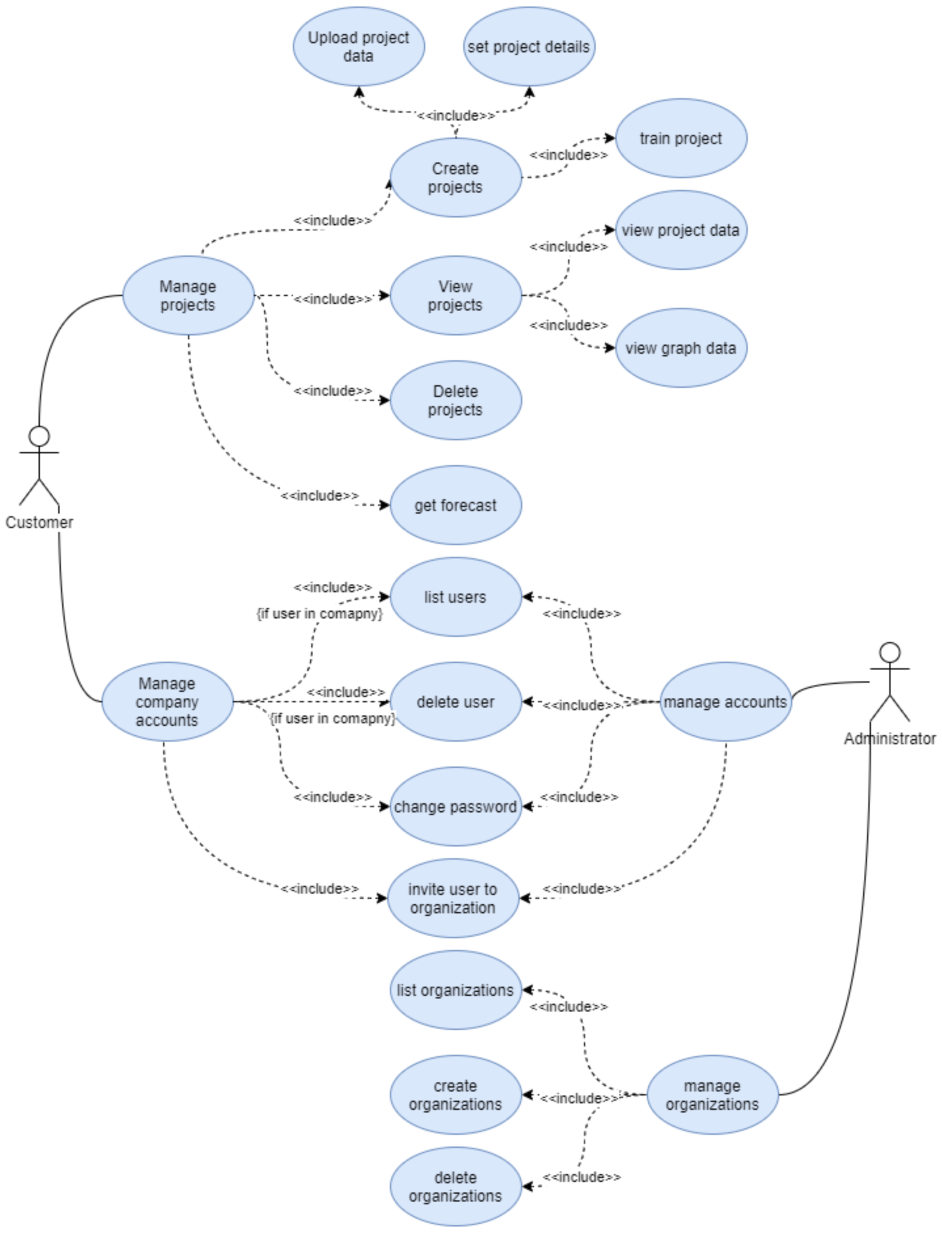


Figure 5.21: Use case diagram

Chapter 6

Experiments

This chapter will present the experiments that were conducted to ensure the desired system functionality and quality. The first section will introduce the experiments created to verify and optimize performance for the systems wind power production forecasts. The second section will focus more broadly on system functionality and quality, and will presents the experiments intended to validate the system requirements presented in section 5.1.

6.1 Transfer Learning Experiments

In order to verify and optimize forecasting performance for new system users, we conducted several experiments with transfer learning on wind power data. The focus of these experiments were to explore the different forecasting methods described in chapter 4. In this section, subsection 6.1.1 to subsection 6.1.5 describes the experimental setup common for all these experiments, while subsection 6.1.6 describes the aims and unique setup for each experiment.

6.1.1 GEFCOM2014 Wind Power Dataset

The *Global Energy Forecasting Competition 2014* (GEFCOM2014) was a time series forecasting competition for a range of energy related forecasting problems, including a wind power forecasting track. The competition findings and data has been published by Hong et al. [21].

The dataset consists of 700 days of hourly power production data and weather forecasts for ten wind farms. The only information given about the wind farm location is that they are located in the same region of the globe, and the power production data has been min-max scaled to anonymize the wind farms.

We choose to use this public wind power dataset for our experiments as they contain data for more wind farms than what TrønderEnergi manages, allowing more extensive testing for transfer learning. The dataset also allows for better reproducibility through being a public dataset, where as TrønderEnergi's wind power data are proprietary.

6.1.2 Data Preprocessing and Feature Engineering

In order to ensure successful transfer, the production data for each wind farm needs to be scaled on a per-wind farm basis. This would be achieved by dividing production by the wind farms max production, however for the GEFCOM2014 data this scaling has already been done.

In our work we choose to not focus on experimenting with feature engineering, but rather opt to use a feature set previously used by TrønderEnergi for their non-transfer wind power forecasting. We use the following feature set for all experiments.

$$F_t = \{p_{t-5:t-1}, p^{\delta}_{t-5:t-1}, ws_{t-5:t+3}, ws^{\delta}_{t-2:t+2}, wd_{t-2:t+2}, wd^{\delta}_{t-2:t+2}\}$$

Where F_t is the feature set used for predicting wind power at timestep t, p_t is the power at timestep t, ws_t is the wind speed forecast at timestep t and wd_t is the wind direction forecast at timestep t. The colon subscript notation specifies a range of timestep values $p_{a:b} = \{p_a, p_{a+1}, ..., p_b\}$ where $a < b$. The superscript δ specifies a differenced value $p^{\delta}_t = p_t - p_{t-1}$.

6.1.3 Validation Strategy

In our experiments we use each wind farm as a simulated target wind farm in turn, with the other nine wind farms acting as potential source wind farms. The average performance across all ten target wind farms is then reported as the final performance.

The testing of each individual target wind farm utilizes a small set of training data from the target wind farm, and a large set of training data from the source wind farms. The model is then tested on a larger set of target data. We use the *Mean Absolute Error* (MAE) error metric for our experiments. While the target training set has to be small, to simulate a wind farm with little available data, the test set can be large in order to give a more accurate performance measurement. Due to the expected correlation between data from different wind farms, it is important to not have an overlap between source training data and target test data to prevent data leakage. The source training data therefore only consists of data from before the target test set.

Due to the use of a small training set for the target wind farm, a greater variance in performance depending on the content of this training set can be expected compared to standard machine learning with a larger training set. We therefore choose to use a sliding window validation strategy, in which the windows of training and test data only partially covers the full dataset. Multiple passes are then performed, in which a model is trained and tested. After each pass, the windows are slid forwards in time, allowing a new set of data to act as the training set. The test set error is averaged across all passes, and reported as the final error. Figure Figure 6.1 illustrates the sliding window validation strategy.

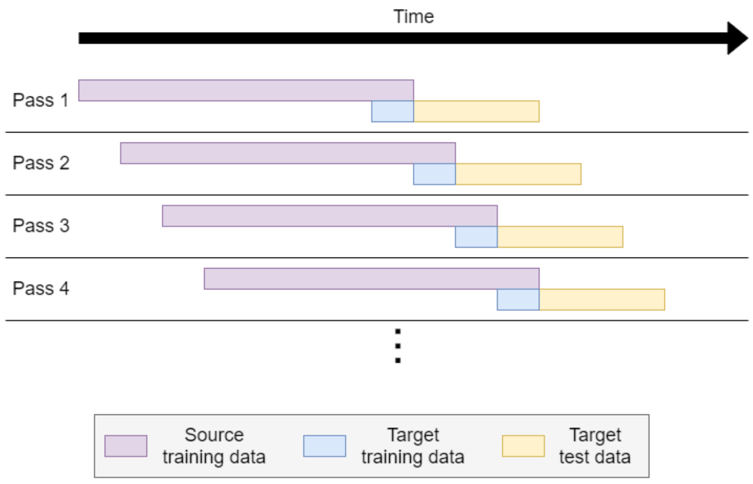


Figure 6.1: Diagram showing the sliding window validation strategy

The total number of data points covered by the validation strategy is given by the formula:

$$d_{total} = max(d_{str}, d_{ttr}) + d_{tte} + (n_{pass} - 1) \cdot d_{shift}$$

Where d_{str} is the size of the source training set, d_{ttr} is the size of the target training set, d_{tte} is the size of the target test set, n_{pass} is the number of passes and d_{shift} is the number of data points that the windows are shifted after each pass.

For our experiments on the GEFCOM2014 datasets, which consists of a total of 16,800 data points, we chose the following parameters for the validation:

$$d_{str} := 8400$$
$$d_{ttr} := 600$$
$$d_{tte} := 1800$$
$$n_{pass} := 12$$
$$d_{shift} := 600$$

This utilizes a source training window of size equal to half the total dataset, allowing the second half of the data to be used for testing. The size of the target train set is 600 data points, corresponding to 25 days of known data for the target wind farm. The total number of data points covered with these parameters matches the total GEFCOM2014 data set size of 16,800 data points.

6.1.4 Baselines

In order evaluate the effectiveness of transfer learning, we compare the performance of different models to two baseline models. A naive baseline, which simply uses the last known wind power production as its prediction, and a GTB model trained only on the target training data, not utilizing any transfer learning.

6.1.5 GTB and TL-GTB Implementation and Parameters

Our implementation of the TL-GTB method described in section 4.1 is built on the CatBoost Python library [9], which provides all the functionality needed for training, slicing and refining GTB ensembles. The CatBoost library comes with automatic selection or well generalizing defaults for all parameters, and claims that it is generally able to achieve great results with default parameters as one of its main features. We therefore run CatBoost training with mostly default parameters, but we set the loss function to our error metric, MAE, and we set the tree depth to 3, which is based on TrønderEnergi's previous practices and results for wind power forecasting. The GTB baseline model is trained for 1000 iterations, while the TL-GTB model is first trained on source data for 1000 iterations, and then refined on target data for 100 iterations.

6.1.6 Experiments

We performed five different experiments, labeled TL-Exp. 1 to TL-Exp. 5. This subsection describes the goal and setup unique to each of these experiments. The result of each experiment will be presented in section 7.1.

TL-Exp. 1: TL-GTB and Source Utilization Effectiveness

This experiment tests the effectiveness of the TL-GTB method, in combination with the three source utilization strategies: single-source transfer, combined-source transfer and transfer ensemble, described in section 4.2. No weighting strategy is employed in this experiment, with equal weighting being given to each source for the two multi-source strategies. The performance of these methods are with each other and the two baseline methods.

TL-Exp. 2: Wind Profile Based Weight Generation and Optimal Cluster Count

This experiment tests the clustering-based wind profile similarity weight generation method described in subsection 4.3.1 This method leaves the cluster count k as a tunable parameter. In order to optimize this weighting strategy we tested a range of cluster counts between 2 clusters and 100 clusters. We tested the resulting performance of using these weights for a weighted TL-GTB transfer ensemble. We set the stability constant s to 0.05 for this test, and target wind farm data in the source training window was used for the weight generation.

TL-Exp. 3: Performance Based Weight Generation and Optimal Decay Factor

This experiment tests the performance based weight generation method described in subsection 4.3.2. This method leaves the exponential decay constant d as a tunable parameter, for which we test 101 values ranging from 0 to 1 at a resolution of 0.01. The generated weights are tested on a TL-GTB transfer ensemble model.

TL-Exp. 4: Multi-Step Forecasting Strategies

This experiment tests the effectiveness of the recursive and the productionless, direct forecasting strategies for multi-step forecasting described in section 4.4. The direct model uses the same features for wind speed and wind direction forecasts as described in subsection 6.1.2, but excludes the wind power features. Both the recursive model and the direct model strategies are tested for TL-GTB transfer ensemble models with equal weights and transferless GTB models. We test and compare these models and multi-step forecasting strategies to the naive baseline for forecasting horizons ranging from 1 to 48 hours.

TL-Exp. 5: Forecasting Performance for Varying Target Data Sizes

This experiment tests the development of forecasting performance as the set of available target training data grows. We compare a TL-GTB transfer ensemble model with equal weights to the naive baseline and the transferless GTB baseline. We test performance for values of the target training set size $d_t tr$ ranging from 100 to 8400.

6.2 System Experiments

This section will cover the experiments used to test the requirements of the system. The purpose of the experiments are to verify if the system has the desired features and qualities. All tests were carried out on the production ready system, to simulate real word usage. When the tests was conducted the system was using a minimum of three nodes, which is the minimum recommended configuration for a Kubernetes cluster in production [30]. However for most of the development period and after delivery the system will be using only one node to save costs. The nodes are implemented using Azure Virtual Machines, and each node has a capacity of 2 vCPUs and 7 GiB of memory. A list of the functional requirements can be seen in Table 5.1 and Table 5.2, while a list of the non-functional requirements can be seen in Table 5.3.

6.2.1 Functional Requirements

Each individual requirement was tested to ensure that the system has the intended functionality. The system was also presented to TrønderEnergi, to ensure that the system was created to their specifications.

6.2.2 Non-functional Requirements

The non-functional requirements had to be tested using different tools and processes. The experiments performed for NFR1-NFR5 were based on the following metrics and environments.

- **NFR1** was tested using the build time and deploy metrics gathered by Azure DevOps.
- **NFR2** was tested using the DevTools in a Chromium based web browser, with a internet connection of 100 Mbit/s.
- **NFR3** was tested using the DevTools in a Chromium based web browser, with a internet connection of 100 Mbit/s.
- **NFR4** was tested on a end-user without any machine learning knowledge.
- **NFR5** was tested by creating a new project and taking the time before the project reached the status ”Ready to forecast”.

The remaining requirements (NFR6-NFR11) required a more specific explanation and experiments for each requirement is presented below.

Scalability of the Entire System (NFR6)

The experiment conducted to test NFR6 was performed using the python load test framework Locust [31]. Locust is a open source framework that can simulate up to millions of concurrent users [31]. For each of the users, Locust can be used to define a set of tasks that will be chosen at random. Each task perform one or more HTTP requests and has a weight, such that some tasks can be ran more often then others. For NFR6, Locust was configured to use up to 500 users, where 5 users joined every second. In addition every user would select a task randomly after 10-20 seconds. All pages were weighted equally, except for the request to create a new project, which was given a weight of one hundredth of the other tasks. This behavior was chosen to simulate how we think real users would use the site. As an example we do not think a user will create a new projects very often.

Scalability of Training (NFR7)

The experiment conducted to test NFR7, utilized Locust, the same python test framework used for NFR6. However for this test, Locust was configure for one task only; initialize training. The test will be conducted in two parts, first the system will be set up to train 10 projects made 10 seconds apart. The first test was concluded when all the projects had the state "Ready to forecast". Before the second test was conducted, the system was given time to scale down to its initial configuration. For the second test the system was set to train 20 projects, made 10 seconds apart, with the same complete condition. Finally the time used by the tests were compared based on the total training time.

Modifiability of the System (NFR8-NFR11)

NFR8-NFR11 are requirements that represent system changes that could be required for the system to stay competitive. That is, they are concerned with the modifiability of the system. The problem with these requirements are that they require the changes to be implemented to get an accurate test result, and some even require special expertise. Performing the actual changes are time consuming and outside the scope of our this project. There exists methods for validating the modifiability of systems, such as Bengtsson et al. [3], however these methods are not a prefect match for our system. Instead we have taken inspiration from Bengtsson et al. [3], and adapted them to our requirements. To evaluate NFR8-NFR11 we would have to estimate the time needed to make the changes for each requirement. This estimate was based on the software architecture and the precises change that would have to be made to the system. This was achieved by presenting an implementation strategy for each requirement. Finally the strategy was evaluated against the requirement.

Chapter 7

Results and Discussion

This chapter presents and discusses the results of the experiments described in chapter 6. section 7.1 presents the results of the experiments on transfer learning, while section 7.2 presents the result of the experiments testing the system requirements.

7.1 Transfer Learning Results

This section presents the results the transfer learning experiments described in subsection 6.1.6. Subsection 7.1.1 to subsection 7.1.5 presents the result of each of the five experiments in turn, while subsection 7.1.6 summarizes and discusses these results.

7.1.1 TL-Exp. 1: TL-GTB and Source Utilization Effectiveness

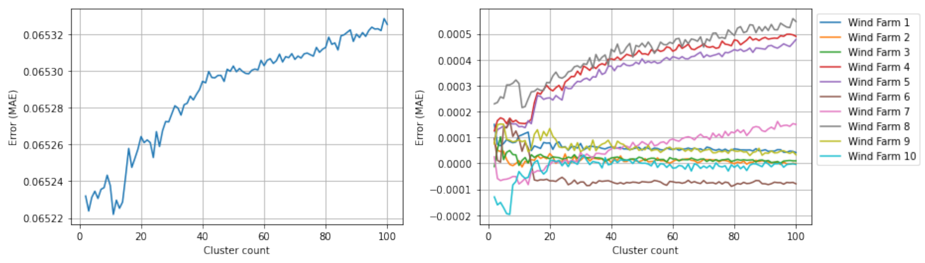
	Naive Baseline	GTB Baseline	Single-Source TL-GTB (Best)	Combined-Source TL-GTB	Ensemble TL-GTB
Wind Farm 1	**0.0628**	0.0726	0.0680	0.0670	0.0663
Wind Farm 2	**0.0546**	0.0630	0.0566	0.0557	0.0553
Wind Farm 3	0.0640	0.0660	0.0620	0.0615	**0.0603**
Wind Farm 4	0.0735	0.0813	0.0755	0.0752	**0.0733**
Wind Farm 5	0.0694	0.0708	0.0675	0.0660	**0.0654**
Wind Farm 6	0.0701	0.0736	0.0688	0.0682	**0.0669**
Wind Farm 7	0.0606	0.0660	0.0612	0.0603	**0.0595**
Wind Farm 8	**0.0662**	0.0783	0.0697	0.0686	0.0685
Wind Farm 9	0.0668	0.0727	0.0681	0.0675	**0.0661**
Wind Farm 10	0.0750	0.0759	0.0712	0.0707	**0.0699**
Mean	0.0663	0.0720	0.0668	0.0661	**0.0652**

Table 7.1: The mean absolute error for TL-GTB with each source utilization strategy compared to baselines for each target wind farm. Lowest error is marked as bold.

Table 7.1 shows the test error achieved by each of the tested methods on each target wind farm. For the sake of conserving space, only the error of best performing single-source TL-GTB model is presented for each wind farm. These results show a notable reduction in error from using transfer learning compared to the transferless GTB model, with the ensemble transfer strategy consistently performing the best out of the three source utilization strategies.

While ensemble TL-GTB outperformed the naive baseline for most target wind farms and on average, it performed poorer than the naive baseline on wind farm 1, 2 and 8. It is however important to note that where as the performance of ensemble TL-GTB improves with more training data, the naive baseline performance remains fixed. TL-Exp. 5 explores this relationship in greater detail.

7.1.2 TL-Exp. 2: Wind Profile Based Weight Generation and Optimal Cluster Count



(a) Average error.

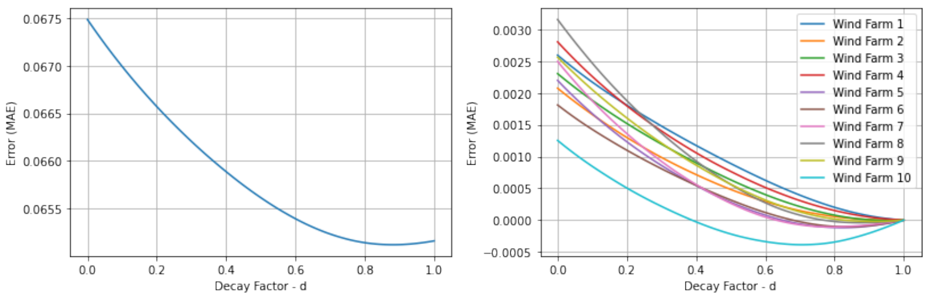
(b) Error deviation from equal weights for each wind farm.

Figure 7.1: Plot of Mean Absolute Error for different cluster counts c.

Figure 7.1 shows the developing error over increasing cluster counts for the cluster-based wind profile similarity weight generation method. The best average performance was achieved with 11 clusters, resulting in an average error of *0.065222*, which is slightly worse than the average error achieved by the previously tested equal weight strategy, which is *0.06516*. Both this difference and the increase in error from higher cluster counts is far too small to hold particular significance.

It is interesting to note how different cluster count impacted different wind farms, with a greater variation early on, then different converging behaviours as cluster count increases. While the error steadily increases for some wind farms as cluster count grows, other wind farms see stable performance or even very slight improvements with increasing cluster counts.

7.1.3 TL-Exp. 3: Performance Based Weight Generation and Optimal Decay Factor



(a) Average error. (b) Error deviation from equal weights for each wind farm.

Figure 7.2: Plot of Mean Absolute Error for different exponential decay factors d.

Figure 7.2 shows the developing error over increasing exponential decay factor d for the performance-based weighting strategy. A value of $d = 0$ corresponds to a single-source utilization strategy, while $d = 1$ corresponds to an equal weight strategy. The best average performance is achieved with the value of d being 0.88, resulting in an average error of *0.06512*, a very slight improvement from the average error achieved by equal weights, which is *0.06516*. This improvement in error is however too small to be of particular significance.

As can be seen in Figure 7.1(b) a value of d lower than 1 does not yield the best performance for every wind farm. Some wind farms have a strictly decreasing error as d increases, while others reach a minimum error at a lower value for d.

7.1.4 TL-Exp. 4: Multi-Step Forecasting Strategies

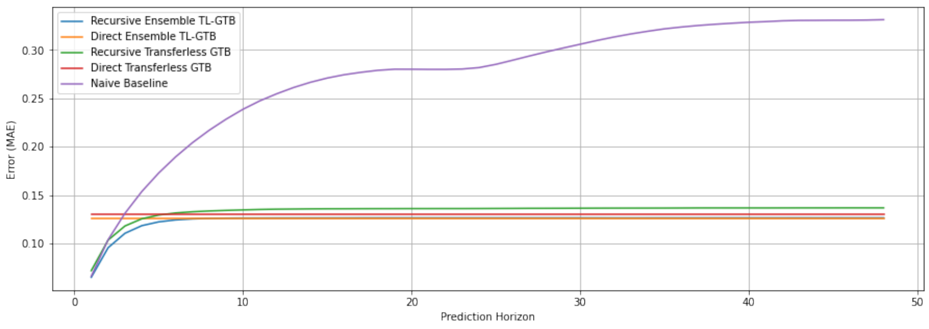


Figure 7.3: Plot of mean absolute error over horizons from 1 to 48.

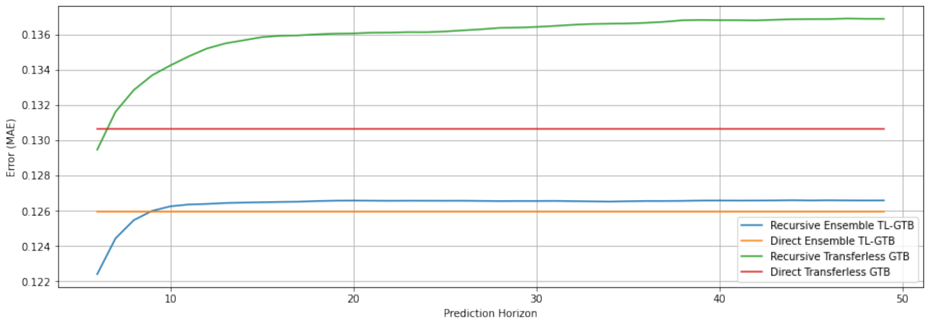


Figure 7.4: Plot of mean absolute error over horizons from 5 to 48, excluding naive baseline.

Figure 7.3 shows the forecasting error from 1 to 48 hour horizons for both the recursive and direct forecasting strategies using TL-GTB and transferless GTB, compared to the naive baseline. Figure 7.4 shows the error from 5 to 48 horizons and excludes the naive baseline for improved distinguishability.

The direct models maintains a fixed error across all horizons. The recursive models errors starts low and increases over the first few horizons, but flattens out. The naive baseline flattens out towards the 24th horizon, but starts increasing again after the 24 hour horizon. This is presumably caused by a daily seasonality in wind power production.

While the recursive TL-GTB model only outperforms the naive baseline slightly for short horizons, the error of the naive baseline increases significantly faster and for significantly longer than the recursive model. The recursive model outperforms the direct TL-GTB model for the first 7 horizons, after which the direct model performs slightly better. After approximately 18 horizons, the recursive TL-GTB model error reaches convergence at a MAE of *0.1266*, approximately 0.5% higher than the direct TL-GTB models MAE of *0.1260*.

When comparing the ensemble TL-GTB models with the transferless GTB models, TL-GTB performs consistently better across all horizons. It is interesting to note that the difference in performance between the recursive and direct models for transferless GTB is significantly greater than for TL-GTB. For transferless GTB the recursive model only outperforms the direct model for 5 horizons. Where as the error of the recursive TL-GTB converges after approximately 18 horizons, the error of the recursive, transferless GTB model keeps raising in later horizons.

7.1.5 TL-Exp. 5: Forecasting Performance for Varying Target Data Sizes

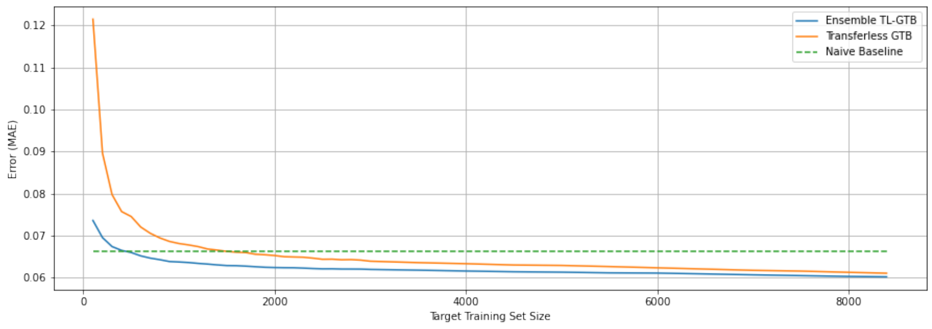


Figure 7.5: Plot of mean absolute error over increasing target training data.

Figure 7.5 shows the single-step forecasting error of ensemble TL-GTB compared to transferless GTB and naive baseline for target training set sizes ranging from 100 to 8400 datapoints. Particularly for smaller target training set sizes, ensemble TL-GTB significantly outperforms transferless GTB, but the difference in performance between these two gradually decreases as target training data increases. Ensemble TL-GTB still outperforms transferless GTB for the largest tested training set size of 8400 datapoints, or 350 days of data, with an MAE of

0.06019 compared to transferless GTB's MAE of *0.06104*. Ensemble TL-GTB bypasses the performance of the naive baseline for single-step forecasts at a target training set size of 500 and higher. The transferless GTB model outperforms the naive baseline at 1500 datapoints and higher.

7.1.6 Summary and Discussion

These experiments show that the TL-GTB method using the ensemble source utilization strategy consistently improves prediction performance compared to not using transfer learning, and allows for particularly very large improvements for wind farms with little available data. TL-Exp. 2 and TL-Exp. 3 shows that our two tested weighting strategies fails to yield significant improvements in forecasting accuracy. While TL-Exp. 5 shows that the naive baseline still outperforms ensemble TL-GTB for very small target training set sizes on single-step forecasting, TL-Exp. 4 also shows that the error of the baseline increases significantly faster over horizons. We therefore expect that ensemble TL-GTB will outperform the naive baseline for longer horizon even with very small amounts of data, however we lack experiments to confirm this.

Based on these results, the model we implemented in out forecasting system is an ensemble TL-GTB model with equal weights, using the recursive multi-step forecasting strategy to provide forecasts from 1 to 48 hour horizons. While TL-Exp. 4 shows that a productionless direct model slightly outperforms the recursive model for higher horizon, we choose not to utilize such a model for higher horizons for two reasons. Firstly having to train a productionless model in addition to a model utilizing production features would double the system training time. Secondly, the error from the recursive model is propagated forwards through horizons, with only a small expected shift in error per horizon. This results in a smooth and realistic looking forecast. For a direct model however, the error is not propagated forwards, but regenerated for each horizon. Using a direct model to create forecasts could result in rapid changes between greatly overestimating one horizon, then greatly underestimating the next, creating a more jagged and unrealistic looking forecast. While this method might have a slightly lower forecasting error, it could easily seem like a significantly poorer forecast to the inexperienced end user. These two reasons, combined with the accuracy gain being fairly small, made us opt for only using a single, recursive model.

7.2 System Results

This section presents and discusses the results of the system experiments presented in section 6.2. The experiments were created to validate the features and qualities of the system.

7.2.1 Functional Requirements

Each of the requirements presented in Table 5.1 and Table 5.2 were implemented into the system. All of the requirements was validated with success. TrønderEnergi was also given a demonstration of the final product, and they were satisfied with the result.

7.2.2 Non-functional Requirements

To validate that the system had the desired system qualities each of the non-functional requirements presented in Table 5.3 was validated based on the setup described in subsection 6.2.2. Many of the non-functional requirements could be tested directly with little configuration, and the results from those experiments are outlined in Table 7.2. However some of the experiments, particularly those related to modifiability and scalability, requires a more in depth discussion and explanation. These experiments are presented in their own sections below.

ID	Test result
NFR1	Each of the services are normally deployed to production within 5 minutes.
NFR2	After having performed 10 predictions without a set date, the average response time was 4.6 seconds.
NFR3	Most pages loads under 500ms, however the project pages averaged at 1.9 seconds over 10 attempts.
NFR4	With the premise of the user having the park information and training data in front of them, it took our one test user 1 minute 1 seconds to complete the park setup.
NFR5	The weather data acquisition and training took 2 minute and 45 seconds for a test park with 16.8k rows.

Table 7.2: The result of the non-functional requirements experiments

Scalability of the Entire System (NFR6)

NFR6 is concerned with the systems ability to scale to accommodate increased user volume. The system was able to serve 500 concurrent users without any issues. During the test Locust performed a total of 3902 requests, including 4 new project requests with an average response time of 295ms seconds during the two minutes 10 seconds long test. Figure 7.6 shows the graphs created by Locust to illustrate how the user load increased with time. While the tests were being executed the Management Service was scaled from three replicas to five replicas, to handle the increasing load. This tests illustrate that the system is able to scale to accommodate peeks in system usage.

Scalability of Training (NFR7)

NFR7 is concerned with the systems ability to scale, such that it can train multiple models at the same time. The first test trained 10 parks in 11 minutes 45 seconds, while the second test trained 20 parks in 15 minutes 5 seconds. For both tests one park was added every 10 seconds, meaning that the last test should use 1 minute 40 seconds more then the first. However in our test, the second test used 3 minutes 20 seconds more then the first test. This means that the second test was not able to train the parks as quickly as the first test.

During the tests, we monitored the number of nodes, and for the first test it scaled the cluster to four nodes, adding one from the default configuration. For the second test it scaled the cluster to seven nodes, adding four nodes. The difference in the number of new nodes, is important to the overall time, as the adding of a single node can take minutes. As such we think it is likely that the time needed to add a new node, is the reason for the longer training time.

Even tough the result did not reach the requirement, it clearly shows that the system is able to scale to accommodate increased demand. A possible solution to achieve the desired result, is to use Azure Container Instance instead of Azure Virtual Machines for Kubernetes nodes. The cluster used for this test did not have access to Container Instances, as such we will leave such a configuration for future work.

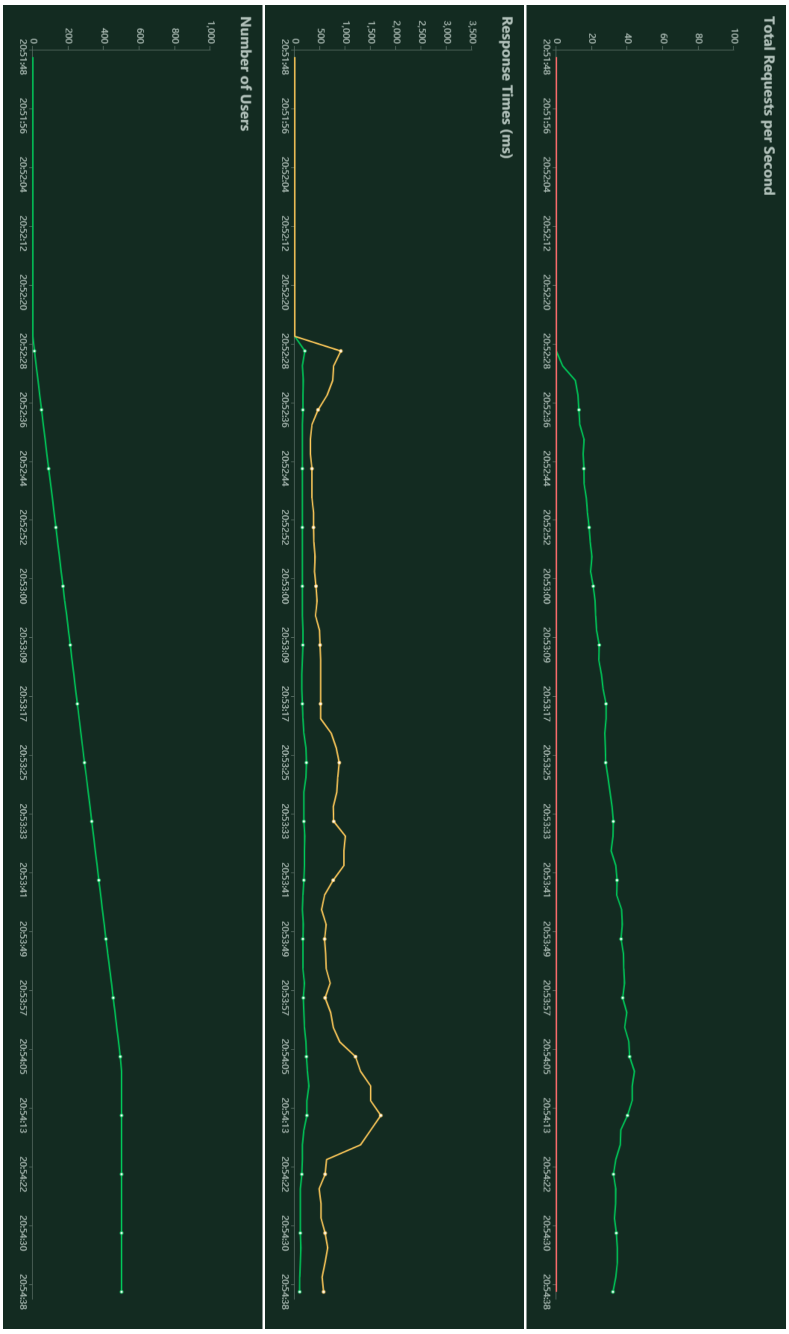


Figure 7.6: Graphs used to visualize the load for NFR6.

Modifiability of the System (NFR8)

NFR8 is directly related to the modifiability of the system, and the following steps have been taken to ensure that a data scientist is able to modify the training used for wind power forecasting.

- The functionality related to wind power project training is created as its own module; the Training Service.
- The code used for the Training Service is hosted in its own GitHub repository.
- The deployment process is automated using CI/CD. The Training Service can be changed without needing any knowledge of how the system is deployed and without any need for manual labor.
- The Training Service comes with a README, that includes the steps needed to get started.
- The usage of docker-compose means that no manual configuration is required to run the Training Service locally, and since it is running in Docker its behavior is independent of the operating system of the user.
- TrønderEnergi can leverage the comprehensive documentation of the Training Service presented in chapter 5.
- The Training Service is using python, the same language used by data scientists at TrønderEnergi.

Some of theses properties can be directly related to the architecture modifiability tactics described in chapter 2. Since the training module is only responsible for training wind power forecasting models, this increases the cohesion and keeps the module size small. The Training Service is minimizing the connection to other services, and connections are only using a few standardized interfaces (Redis, file storage and the share PostgreSQL database), this reduces coupling and technical debt. Even though these steps are not a guarantee that the training module is easy to modify, the steps taken should make it easier for TrønderEnergi to modify the training progress used for wind power forecasting.

Modifiability of the System (NFR9)

NFR8 is concerned with upgrading the system to support multiple time series related tasks. The system is designed such that it should be possible to facilitate for other time series related problems, than just wind power forecasting. An implementation of a new problem type could be achieved with the following steps.

1. The current training process is implemented as an independent service. This means that new training processes can be added as independent services as well. The Management Service would then deploy different Kubernetes Jobs based on the project type.
2. The project table could be extended to support multiple project types by adding a type field and any other metadata information to the project table, as well a corresponding data table for each type. This would be implemented by adding a new Django model for the data table, and by editing the existing projects model
3. The user interface would have to be updated on the main project page, the create project page and the project page. These pages would have to be modified to show information specific to each project type.
4. The Cronjobs for fetching weather and making forecasts would have to be updated to support multiple projects, their behavior should depend on the project type.
5. Inference Service would have to be updated to support forecasts made by the new training service.

Assuming that the new time series problem utilized the same project page layout as for wind power forecasting, we estimate that a project type could be implemented in about 70 hours, excluding the time needed to created the training module. This includes 25 hours to fix the user interface, 30 hours to configure the Management Service for the new training module including tests, 10 hours to update the Inference Service and 5 hours to configure CI/CD for the new training module. This is well within the 100 hours requirement. However this estimate could differ based on the data that should be displayed, and whether or not the new project has meta data that is very different from the current meta data.

Modifiability of the System (NFR10)

NFR10 is concerned with extending the system to support organizational permissions for users. This functionality is intended to limit users ability to invite and delete users, such that only a select group of users with elevated permissions would have this access. The easiest implementation of this feature, would be to add a boolean field is_organizational_admin to the user model. This field should be checked when the user tries to invite or delete a user. This features would also require extensive testing as it is vital to the security of the application. We estimate that it would take about 2 hours to implement the functionality and another 2 to tests it, as such it should be possible to archive within 10 hours. For a more advance permission system, it might be feasible to add specific organizational permissions to each user, however this is out of the scope of this requirements.

Modifiability of the System (NFR11)

NFR11 is concerned with platform lock-in, and the ability to change cloud provider. Since the system is using Kubernetes, it can be ported to any of the major cloud platforms without any modifications to the compute platform. There are however a few aspects of the system that would have to be changed. Firstly the CI/CD pipelines in Azure DevOps would have to be move to a similar service on the new platform. Secondly the current azure file storage would have to be moved to a similar service. Thirdly the PostgreSQL database would have to be transferred to a new PostgreSQL database on the new platform. We estimate that this process should be achievable within 30 hours, well within the 50 hour limit.

Chapter 8

Evaluation and Future Work

In this chapter we will evaluate the system implementation in regard to our hypothesis and research questions presented in the introduction. Based on the findings we will make suggestions for future work.

8.1 Evaluation

The purpose of our research was to identify if an automatic system for wind power forecasting could be designed as a competitive alternative to task specific implementations and AutoML. In collaboration with TrønderEnergi we concluded that such a service would have to be easier to use than task specific implementations and achieve better forecasts then AutoML. The service could have an additional competitive advantage if we could leverage the collected data in the system to improve forecasts for all wind farms, particular wind farms with little available data. If such a system could be implemented, it should also be easy to modify such that it could evolve to a larger time series based forecasting system. To achieve this goal, we proposed the following hypothesis:

HYP: *An automatic, scalable machine learning service can be developed for wind power production forecasting for wind farms with little available data.*

To enable the verification of this hypothesis, three research questions were defined.

RQ1: *How can transfer learning be applied to improve wind power forecasting performance?*

We have shown through several experiments that the use of transfer learning with gradient tree boosting gives notable improvement in wind power production forecasting for a wide range of target data set sizes and forecasting horizons. These results are based on testing on 10 different open wind farm datasets, using a sliding window validation method, achieving a high degree of statistical significance in the result.

While we show that use of transfer learning can be a significant benefit for wind power forecasting, we have only tested transfer learning for gradient tree boosting. Other alternative approaches to transfer learning, for example neural network based approaches, might prove even more successful.

RQ2: *How can a system be designed to automate the use of transfer learning for wind power forecasting?*

We have demonstrated such a system through experimental development, yielding a functional, fully automatic system for wind power forecasting, which implements a transfer learning based machine learning model. The machine learning method has been designed based on a standard configuration of parameters and features, supporting the training of new models for new wind farms without any wind farm specific adjustments or overhead.

This automatic system does however pose some risks, which have not been sufficiently investigated. subsection 3.3.1 describes the technical debt cost of increasing data dependencies, and how unstable data dependencies might result in unexpected behavior of machine learning models. As the number of wind farms in our system grows, more and more candidate source wind farms are available for training models for new target wind farms. While this can be a valuable resource for improved forecasting performance, it also comes at the cost of growing data dependencies for models. In order to prevent training time for new wind farms to increase as more wind farms are added to the system, we limited the set of source wind farms to the 10 wind farms with shortest geographical distance to the new target wind farm. This approach does however mean that as time goes on, the set of selected source wind farms for existing wind farms in the system can change, having potentially unpredictable and undesirable effects on model performance. Due to not having wind power data for sufficiently many wind farms and the geographical location of wind farms from the GEFCOM2014 wind power dataset being unknown, we have not been able to conduct a thorough analysis of these effects.

RQ3: *How can a system for wind power production forecasting utilizing transfer learning be designed to support modifiability and scalability?*

Several steps have been taken to design a system that is highly modifiable and scalable. For modifiability this includes using a microservice architecture, reduction of technical debt, a high degree of automation and using platforms and programming languages familiar to TrønderEnergi. The use of microservices divides the system into small independent services with high cohesion and low coupling, in line with many of the architecture tactics for modifiability. During the entire development process, reducing technical debt was critical, and in particular architecture technical debt. To the best of our ability we tried to create robust generic solutions that would make it easier to change the services later. This includes using Redis, PostgreSQL and file storage as the only communication channel for all services, creating a common database manager library to be used for communication with the database and utilizing common libraries for our machine learning module. The automation of testing using GitHub Actions and deploying applications using Azure DevOps means that any developer can update the platform without any knowledge of how the system is deployed. Finally the services are made specific to support TrønderEnergi as they already use Python and Kubernetes for their own projects, thus minimizing the training needed to understand the system.

In addition to the general decisions we made to support modifiability, we also created a set of experiments to test the non-functional requirements related to modifiability(NFR8-NFR11). These experiments tests how easy it is to make certain changes to the system, and the experiments are intended to support the changes that we estimate are likely to occrue. Based on the experiments the system should be able to support the future needs of TrønderEnergi, but they are only estimates. The true modifiability of the system will not be established before the changes are actually implemented, however we are confident that the design decisions we made should positively impact the modifiability of the system.

Steps has also been taken to ensure the scalability of the system. The primary factors include the use of small services, container orchestration and cloud services. Small services allows us to scale each service individually, for better resource utilization. The container orchestration tool, in our case Kubernetes allows us to automatically vertically scale the services based on demand and finally the use of cloud services gives the system the ability to add computer resources almost instantly.

To quantify the exact performance metrics we wanted for the system we created

NFR6 and NFR7. By setting up experiments utilizing Locust we were able to confirm that the system could scale to accommodate up to 500 concurrent users and that training multiple models would not significantly affect training time. We are satisfied with the scalability of the system, however there is room for improvement. We will therefore suggest researching other compute units as part of future work.

Based on the work conducted to answer the research questions, we have created a scalable system that automates the process of performing wind power forecasts, even for wind farms with little available data. In conclusion this verifies our hypothesis.

8.2 Future Work

The conducted research has shown that it is possible to create an automatic wind power forecasting system, that support minimal data. As part of this process we found aspects of our research that requires further work. This section will present these problems.

While our tested method of transfer learning, TL-GTB, gave notable improvements in forecasting accuracy compared to transferless GTB, there exists many other approaches to transfer learning. Other methods, such as neural network pre-training and refinement, should be tested and compared to TL-GTB.

While we investigated the effectiveness of TL-GTB on multi-step prediction and the effectiveness of TL-GTB on single-step prediction for varying target training data sizes, we never combined the two. Our results would indicate that for very small quantities of target training data, the naive baseline would outperform TL-GTB for very short horizons, but not for longer horizons. This relationship should be confirmed and investigated further.

Investigating the effect of different feature engineerings on the effectiveness of transfer learning is also relevant for further research. Some feature preprocessing that might not prove effective for transferless learning, might help improving similarity between features across sources and help generalization and transferability of learned knowledge. The only such pre-processing we perform is min-max normalizing the production of each wind farm independently of each other.

While we tested the impact of varying amounts of target data, we did not conduct any experiments on varying quantities of source data. The impact of both varying number of source wind farms and varying quantities of data for each source wind farm should be investigated. Particularly given that source wind farms with significantly less data than the other available source wind farms might hurt the effectiveness of transfer learning, and should be excluded.

We employ a source pre-selection strategy of using the 10 candidate source wind farms with shortest geographical distance to the target wind farm as source wind farms. The effect of this and other computationally efficient source pre-selection strategies, for example data quantity, should be investigated.

For our research we deployed the system to a scalable Kubernetes cluster that utilized general purpose virtual machines with 2 vCPUs and 7 GiB of memory. To further improve training time and training capability other compute units should also be tested. This includes testing GPU virtual machines to reduce training time and potentially use Azure Container Instances to evade provisioning.

Further procedures and tests should be researched and implemented to ensure the stability of the system in production. Some steps towards a reliable system has been implemented, such as using a Kubernetes cluster with minimum 3 nodes, logging in all services, and automated testing for the Management Service. In particular additional steps should be taken to ensure the reliability of the Training Service, for instance by calculating a tests score based on the process presented in subsection 3.3.2.

Currently the system will only train a new model when a new project is created or if the user triggers a retrain. As such the system is not leveraging the full potential of its gowning database. For the system to fully automatically make use of this data, research must be conducted to find how often its beneficial to retrain the models.

A primary factor related to the modifiability of the system, is the possibility of expanding the system to support other time series related problems. However, research must be conducted into the business feasibility of offering predictions for other time series related problems. In addition research should be conducted into the possibility of using transfer learning cross-domain for time series problems, to identify if the system can leverage data from a multi purpose time series service.

Bibliography

[1] Alpiq (2020). Renewable energy management services. `https://www.alpiq.com/energy-solutions/renewable-energy-management/renewable-energy-management-services`. Accessed: 2020-05-28.

[2] Bass, L., Clements, P., and Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley Professional.

[3] Bengtsson, P., Lassing, N., Bosch, J., and van Vliet, H. (2004). Architecture-level modifiability analysis (alma). *Journal of Systems and Software*, 69(1-2):129–147.

[4] Bondi, A. B. (2000). Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, pages 195–203.

[5] Breck, E., Cai, S., Nielsen, E., Salib, M., and Sculley, D. (2017). The ml test score: A rubric for ml production readiness and technical debt reduction. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1123–1132. IEEE.

[6] Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., et al. (2010). Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 47–52. ACM.

[7] Cai, L., Gu, J., Ma, J., and Jin, Z. (2019). Probabilistic wind power forecasting approach via instance-based transfer learning embedded gradient boosting decision trees. *Energies*, 12(1):159.

[8] Caruana, R. (1997). Multitask learning. *Mach. Learn.*, 28(1):41–75.

[9] CatBoost (2020). Catboost - open-source gradient boosting library. `https://catboost.ai/`. Accessed: 2020-06-07.

[10] Conwx (2020). Accurate power forecasts now! `https://conwx.com`. Accessed: 2020-06-07.

[11] de Toledo, S. S., Martini, A., Przybyszewska, A., and Sjøberg, D. I. (2019). Architectural technical debt in microservices: A case study in a large company. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*, pages 78–87. IEEE.

[12] Dekker, M. D. (2016). 4 + 1 architectural view model. `https://commons.wikimedia.org/wiki/File:4+1_Architectural_View_Model.svg`. Accessed: 2020-05-12.

[13] Django (2019). Documentation. `https://docs.djangoproject.com/en/3.0/faq/general/`. Accessed: 2020-04-24.

[14] Docker (2019). Docker. https://www.docker.com.

[15] Dorogush, A. V., Ershov, V., and Gulin, A. (2018). Catboost: gradient boosting with categorical features support. *CoRR*, abs/1810.11363.

[16] Ebert, C., Gallardo, G., Hernantes, J., and Serrano, N. (2016). Devops. *Ieee Software*, 33(3):94–100.

[17] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., and Hutter, F. (2019). Auto-sklearn: Efficient and robust automated machine learning. In *Automated Machine Learning*, pages 113–134. Springer.

[18] Foley, A. M., Leahy, P. G., Marvuglia, A., and McKeogh, E. J. (2012). Current methods and advances in forecasting of wind power generation. *Renewable Energy*, 37(1):1–8.

[19] GitHub (2020). Kubernetes. `https://github.com/kubernetes/kubernetes`. Accessed: 2020-05-28.

[20] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

[21] Hong, T., Pinson, P., Fan, S., Zareipour, H., Troccoli, A., and Hyndman, R. J. (2016). Probabilistic energy forecasting: Global energy forecasting competition 2014 and beyond. *International Journal of Forecasting*, 32(3):896 – 913.

[22] Hu, Q., Zhang, R., and Zhou, Y. (2016). Transfer learning for short-term wind speed prediction with deep neural networks. *Renewable Energy*, 85:83 – 95.

[23] Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). Automated machine learning-methods, systems, challenges.

[24] Hyndman, R. and Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

[25] ISO/IEC 25010 (2011). ISO/IEC 25010:2011, systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models.

[26] Kruchten, P. (1995). The 4+1 view model of architecture. *IEEE Softw.*, 12(6):42–50.

[27] Kruchten, P., Nord, R. L., and Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21.

[28] Kruchten, P., Nord, R. L., Ozkaya, I., and Falessi, D. (2013). Technical debt: Towards a crisper definition report on the 4th international workshop on managing technical debt. *SIGSOFT Softw. Eng. Notes*, 38(5):51–54.

[29] Kubernetes (2020a). Production-grade container orchestration. `https://kubernetes.io`. Accessed: 2020-05-27.

[30] Kubernetes (2020b). Using minikube to create a cluster. `https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/`. Accessed: 2020-06-04.

[31] Locust (2020). An open source load testing tool. `https://locust.io`. Accessed: 2020-06-04.

[32] Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2018). The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802 – 808.

[33] Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

[34] Microsoft (2019). What is azure container instances? `https://docs.microsoft.com/en-us/azure/container-instances/container-instances-overview`. Accessed: 2020-05-28.

[35] Microsoft (2020a). App service overview. `https://docs.microsoft.com/en-us/azure/app-service/overview`. Accessed: 2020-05-28.

[36] Microsoft (2020b). An introduction to azure functions. `https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview`. Accessed: 2020-05-28.

[37] Microsoft (2020c). Overview of azure service fabric. `https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview`. Accessed: 2020-05-28.

[38] Montero-Manso, P., Athanasopoulos, G., Hyndman, R. J., and Talagala, T. S. (2020). Fforma: Feature-based forecast model averaging. *International Journal of Forecasting*, 36(1):86 – 92. M4 Competition.

[39] Mouat, A. (2015). *Using Docker: Developing and Deploying Software with Containers.* " O'Reilly Media, Inc.".

[40] Newman, S. (2015). *Building microservices: designing fine-grained systems.* " O'Reilly Media, Inc.".

[41] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

[42] Powel (2020). Software for sustainable growth. `https://www.powel.no`. Accessed: 2020-05-28.

[43] Pérez, A., Moltó, G., Caballer, M., and Calatrava, A. (2018). Serverless computing for container-based architectures. *Future Generation Computer Systems*, 83:50 – 59.

[44] Qureshi, A. S. and Khan, A. (2019). Adaptive transfer learning in deep neural networks: Wind power prediction using knowledge transfer from region to region and between different task domains. *Computational Intelligence*, 35(4):1088–1112.

[45] Qureshi, A. S., Khan, A., Zameer, A., and Usman, A. (2017). Wind power prediction using deep neural network based meta regression and transfer learning. *Applied Soft Computing*, 58:742 – 755.

[46] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[47] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., and Young, M. (2014). Machine learning: The high interest credit card of technical debt.

[48] Slawek Smyl, Jai Ranganathan, A. P. (2018). M4 forecasting competition: Introducing a new hybrid es-rnn model. `https://eng.uber.com/m4-forecasting-competition/`. Accessed: 2020-06-07.

[49] Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., and Liu, C. (2018). A survey on deep transfer learning. *CoRR*, abs/1808.01974.

[50] Tasnim, S., Rahman, A., Oo, A. M. T., and Haque, M. E. (2018). Wind power prediction in new stations based on knowledge of existing stations: A cluster based multi source domain adaptation approach. *Knowledge-Based Systems*, 145:15 – 24.

[51] Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., and Darrell, T. (2014). Deep domain confusion: Maximizing for domain invariance. *CoRR*, abs/1412.3474.

[52] WindSim (2019). Power forecasting. `http://www.windsim.com/power-forecasting.aspx`. Accessed: 2020-05-27.

Appendix

A Application Screenshots

This section contains a set of screenshots from the system. The screenshots are intended to show how the functionality was implemented. The screenshots are divided into sections based on three main services.

A.1 Management Service

The management service is the largest service, responsible for project and user management. The screenshots are divided into authentication, list of projects that highlights the features of the project list, creation of new projects, the project page and administration.

Authentication

Figure 9.1 shows the login screen. A user is required to enter their email and password to authenticate. If the user for any reason have forgotten their password, an email can be sent to their email to reset it. This is done by clicking the forgot password link, the user will then be redirected to the screen shown in Figure 9.2. On submission the user will be presented with the view seen in Figure 9.3. If the email exists in the system a email similar to Figure 9.4 will be sent to the user's email.

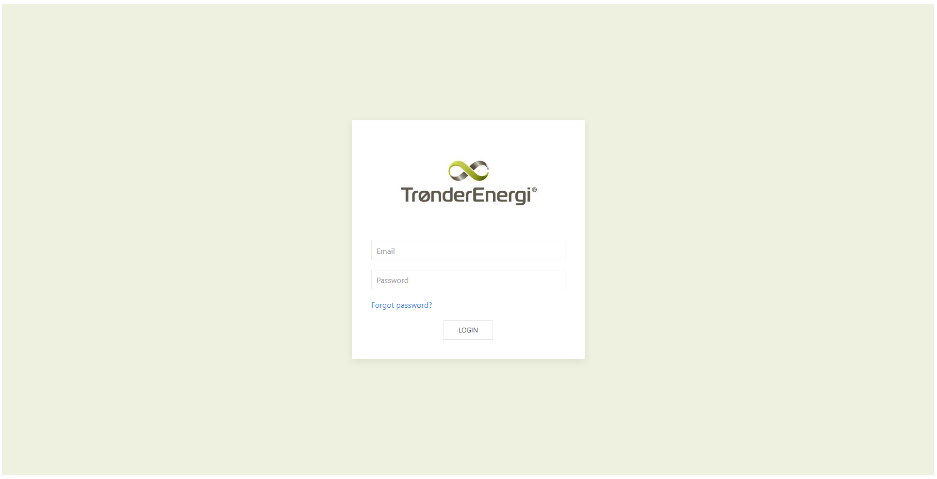
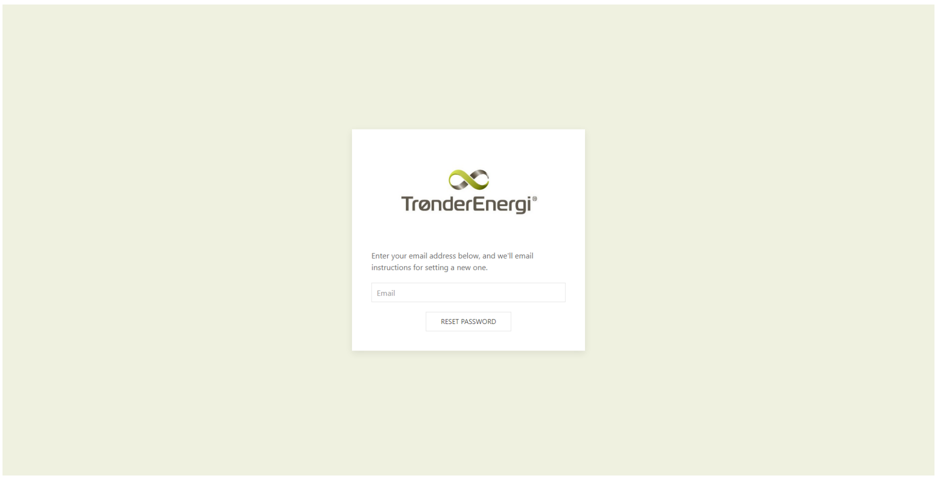
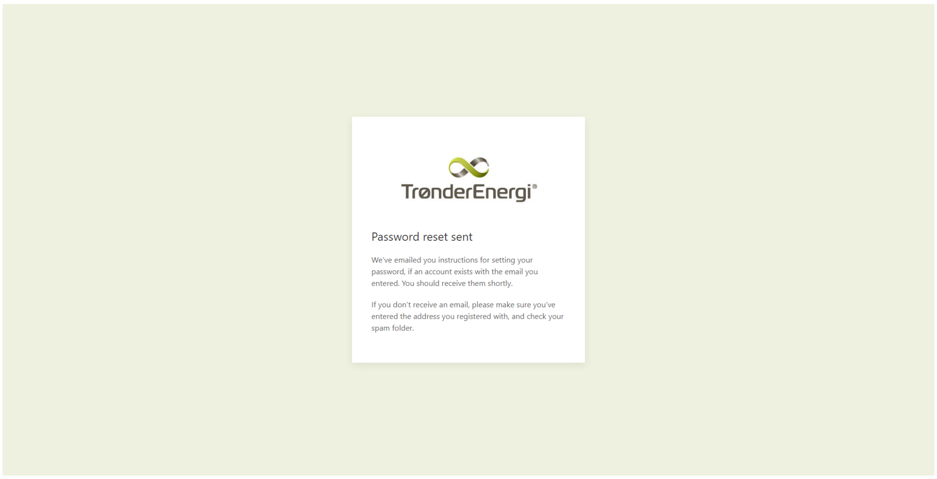


Figure 9.1: Screenshot - login



Enter your email address below, and we'll email instructions for setting a new one.

Figure 9.2: Screenshot - forgot password page



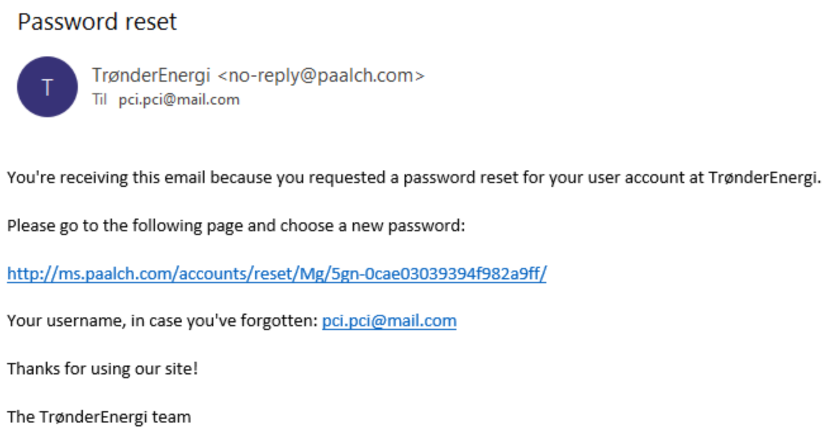
TrønderEnergi®

Password reset sent

We've emailed you instructions for setting your password, if an account exists with the email you entered. You should receive them shortly.

If you don't receive an email, please make sure you've entered the address you registered with, and check your spam folder.

Figure 9.3: Screenshot - reset password confirmation



Password reset

T TrønderEnergi <no-reply@paalch.com>
Til pci.pci@mail.com

You're receiving this email because you requested a password reset for your user account at TrønderEnergi.

Please go to the following page and choose a new password:

http://ms.paalch.com/accounts/reset/Mg/5gn-0cae03039394f982a9ff/

Your username, in case you've forgotten: pci.pci@mail.com

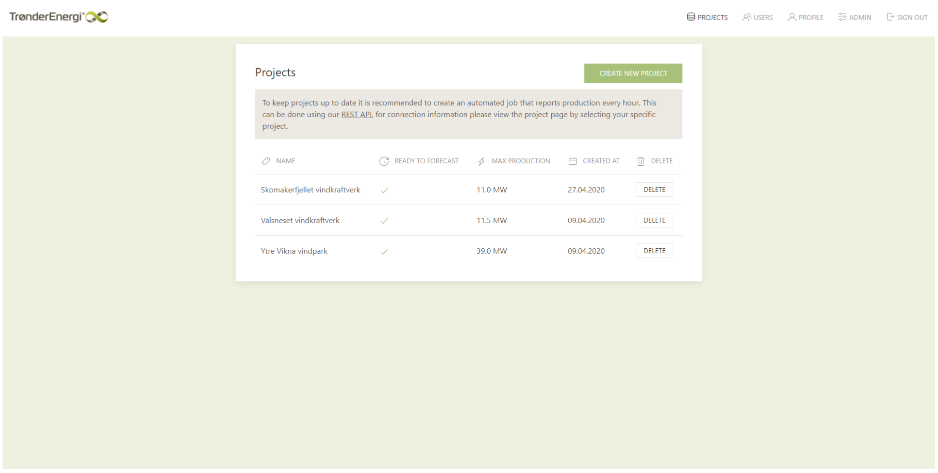
Thanks for using our site!

The TrønderEnergi team

Figure 9.4: Screenshot - reset password email

List of projects

Figure 9.5 show the screen the user will be presented with after login. The page contain a list of the projects owned by the organization of the user. The user got full access to delete, view and create new projects. If the user press the delete button, the user will be greeted with a confirmation modal as seen in Figure 9.6. New projects can be created using the button located on the top right in the main container.

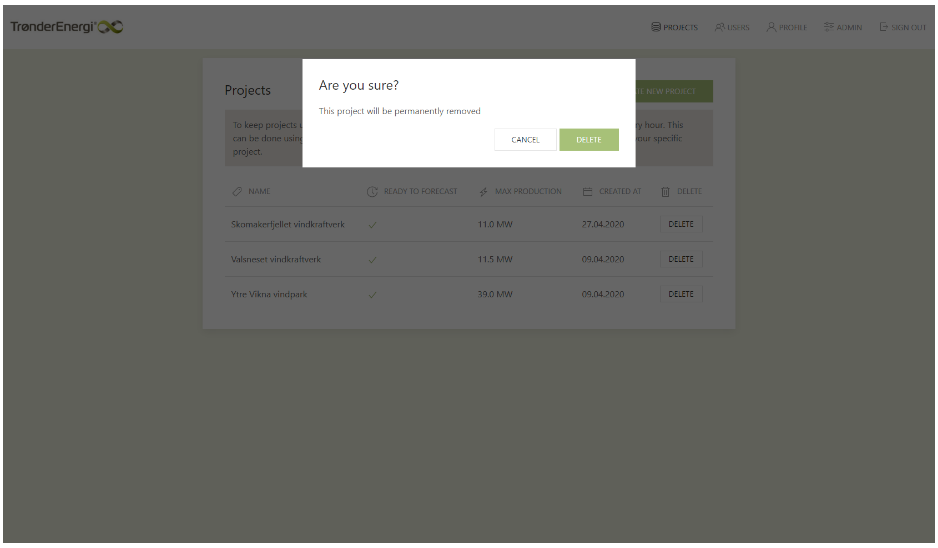


Projects

To keep projects up to date it is recommended to create an automated job that reports production every hour. This can be done using our REST API, for connection information please view the project page by selecting your specific project.

NAME	READY TO FORECAST	MAX PRODUCTION	CREATED AT	DELETE
Skomakerfjellet vindkraftverk	✓	11.0 MW	27.04.2020	DELETE
Valsneset vindkraftverk	✓	11.5 MW	09.04.2020	DELETE
Ytre Vikna vindpark	✓	39.0 MW	09.04.2020	DELETE

Figure 9.5: Screenshot - main page/list of projects



Are you sure?

This project will be permanently removed

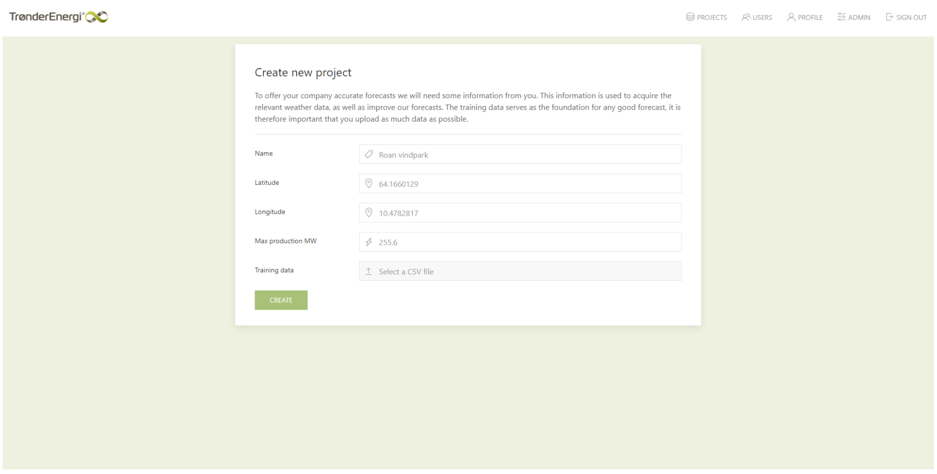
Projects

NAME	READY TO FORECAST	MAX PRODUCTION	CREATED AT	DELETE
Skomakerfjellet vindkraftverk	✓	11.0 MW	27.04.2020	DELETE
Valsneset vindkraftverk	✓	11.5 MW	09.04.2020	DELETE
Ytre Vikna vindpark	✓	39.0 MW	09.04.2020	DELETE

Figure 9.6: Screenshot - delete project

Create new project

When pressing the "create new project" button the user will be redirected to the screen shown on Figure 9.7. It shows the fields that are required to setup a new projects, with placeholders to help the user use the correct format. When filled out, as shown in Figure 9.8, the user may press the submit button, such that training can start. The user will then be shown a loading screen as can be seen in Figure 9.9. As soon as the project is created and the model data is uploaded the user will be redirected to the screen seen in Figure 9.10. The user will then be updated on the progress on left side of the screen. The information will update automatically and includes many steps, Figure 9.11 and Figure 9.12 are examples of such steps. Finally when the training is completed, the user will be shown Figure 9.13.

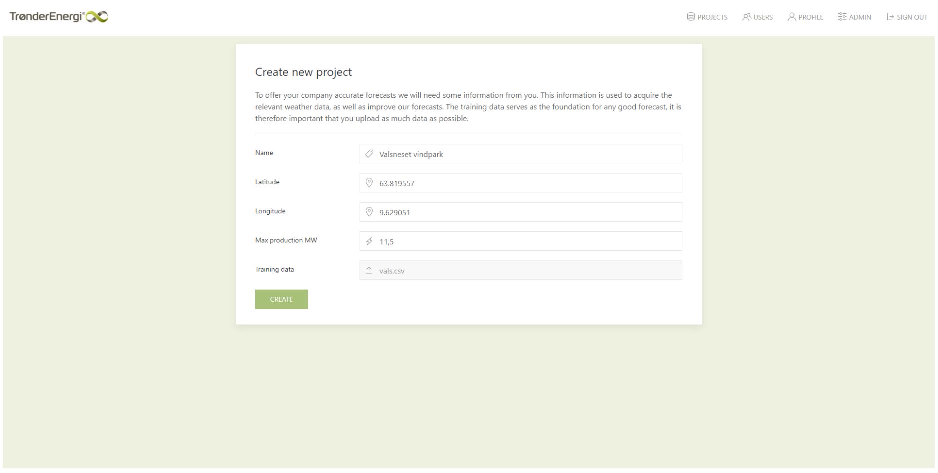


Create new project

To offer your company accurate forecasts we will need some information from you. This information is used to acquire the relevant weather data, as well as improve our forecasts. The training data serves as the foundation for any good forecast, it is therefore important that you upload as much data as possible.

Name	Roan vindpark
Latitude	64.1660129
Longitude	10.4782817
Max production MW	255.6
Training data	Select a CSV file

Figure 9.7: Screenshot - create new project

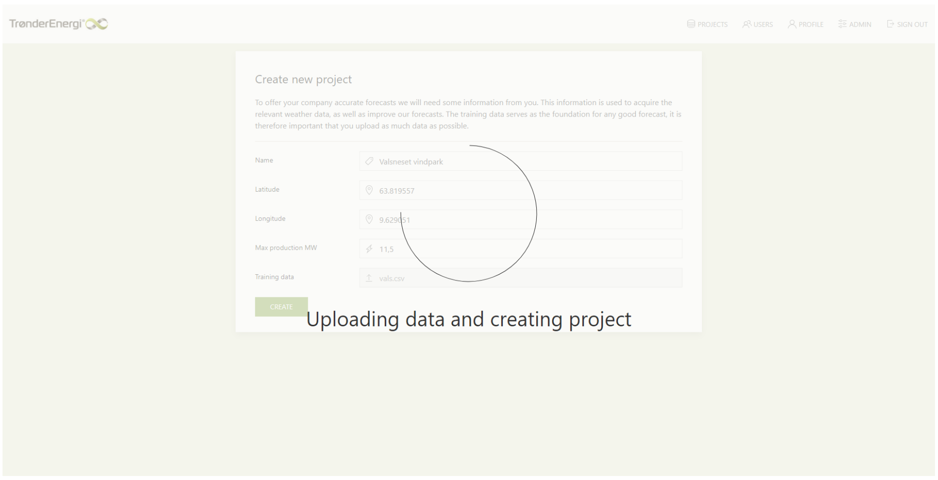


Create new project

To offer your company accurate forecasts we will need some information from you. This information is used to acquire the relevant weather data, as well as improve our forecasts. The training data serves as the foundation for any good forecast, it is therefore important that you upload as much data as possible.

Name	Valsneset vindpark
Latitude	63.819557
Longitude	9.629051
Max production MW	11,5
Training data	vals.csv

Figure 9.8: Screenshot - create new project - including content

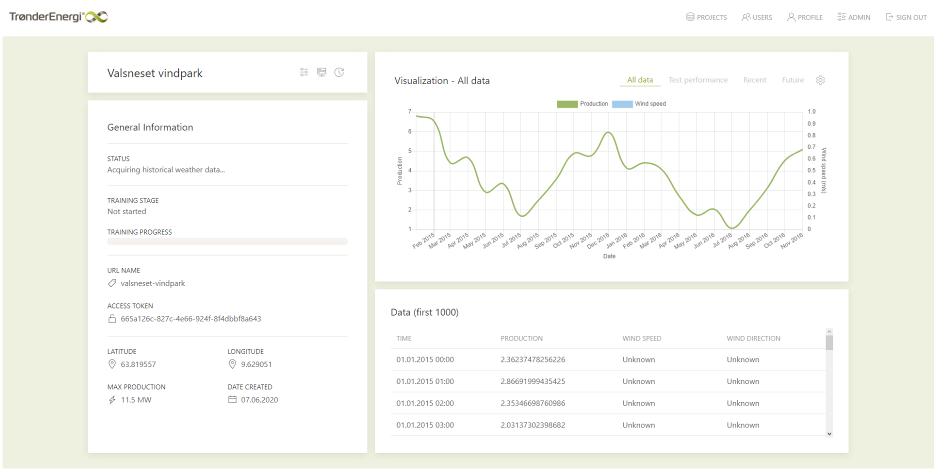


Create new project

To offer your company accurate forecasts we will need some information from you. This information is used to acquire the relevant weather data, as well as improve our forecasts. The training data serves as the foundation for any good forecast, it is therefore important that you upload as much data as possible.

Name	Valsneset vindpark
Latitude	63.819557
Longitude	9.629051
Max production MW	11,5
Training data	vals.csv

Figure 9.9: Screenshot - uploading training data and creating project



Valsneset vindpark

General Information

STATUS
Acquiring historical weather data...

TRAINING STAGE
Not started

TRAINING PROGRESS

URL NAME
valsneset-vindpark

ACCESS TOKEN
665a126c-827c-4e66-924f-8f4dbbf8a643

LATITUDE	LONGITUDE
63.819557	9.629051

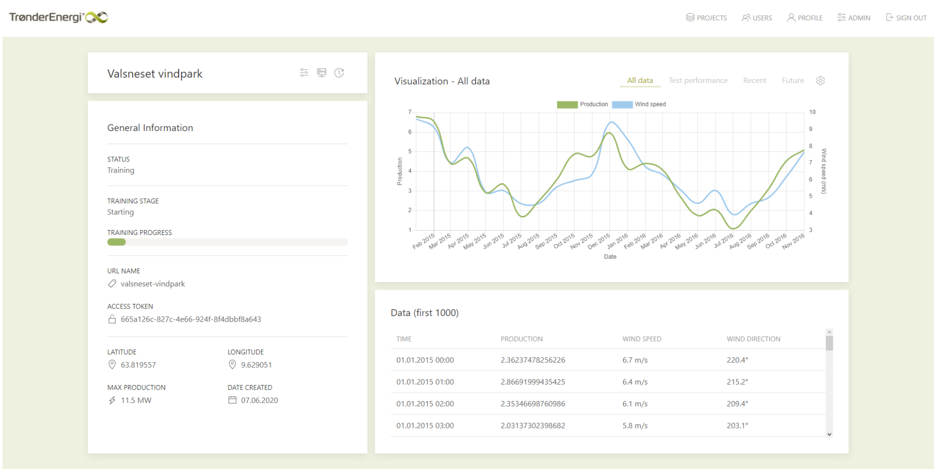
MAX PRODUCTION	DATE CREATED
11.5 MW	07.06.2020

Visualization - All data

Data (first 1000)

TIME	PRODUCTION	WIND SPEED	WIND DIRECTION
01.01.2015 00:00	2.36237478256226	Unknown	Unknown
01.01.2015 01:00	2.86691999435425	Unknown	Unknown
01.01.2015 02:00	2.35346698760986	Unknown	Unknown
01.01.2015 03:00	2.03137302398682	Unknown	Unknown

Figure 9.10: Screenshot - project acquiring weather data



Valsneset vindpark

General Information

STATUS
Training

TRAINING STAGE
Starting

TRAINING PROGRESS

URL NAME
valsneset-vindpark

ACCESS TOKEN
665a126c-827c-4e66-924f-8f4dbbf8a643

LATITUDE	LONGITUDE
63.819557	9.629051

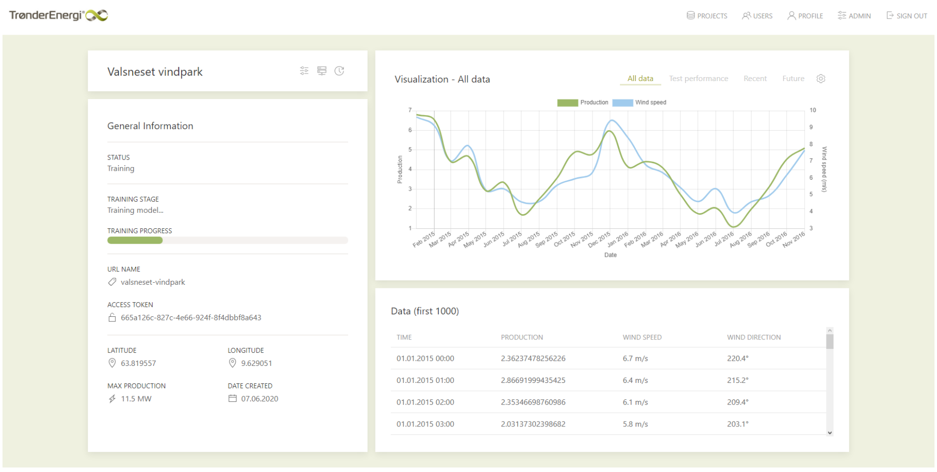
MAX PRODUCTION	DATE CREATED
11.5 MW	07.06.2020

Visualization - All data

Data (first 1000)

TIME	PRODUCTION	WIND SPEED	WIND DIRECTION
01.01.2015 00:00	2.36237478256226	6.7 m/s	220.4°
01.01.2015 01:00	2.86691999435425	6.4 m/s	215.2°
01.01.2015 02:00	2.35346698760986	6.1 m/s	209.4°
01.01.2015 03:00	2.03137302398682	5.8 m/s	203.1°

Figure 9.11: Screenshot - starting training



Valsneset vindpark

General Information

STATUS
Training

TRAINING STAGE
Training model...

TRAINING PROGRESS

URL NAME
valsneset-vindpark

ACCESS TOKEN
665a126c-827c-4e66-924f-8f4dbbf8a643

LATITUDE
63.819557

LONGITUDE
9.629051

MAX PRODUCTION
11.5 MW

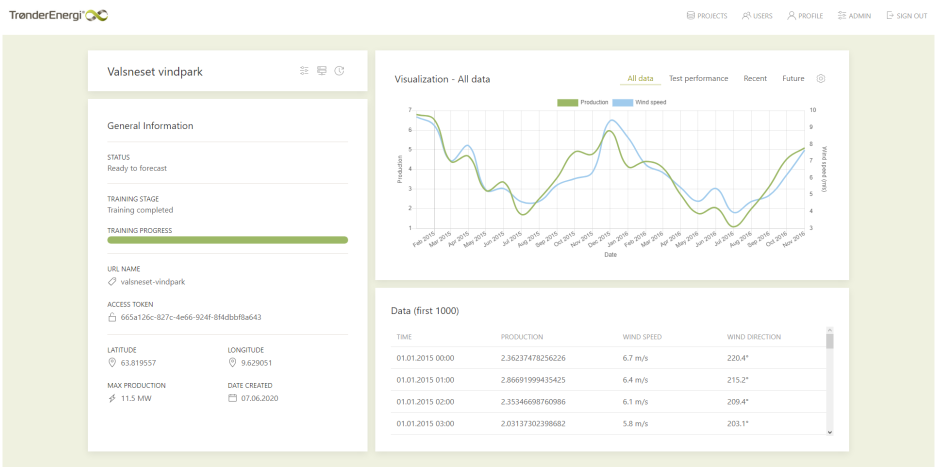
DATE CREATED
07.06.2020

Visualization - All data

Data (first 1000)

TIME	PRODUCTION	WIND SPEED	WIND DIRECTION
01.01.2015 00:00	2.36237478256226	6.7 m/s	220.4°
01.01.2015 01:00	2.86691999435425	6.4 m/s	215.2°
01.01.2015 02:00	2.35346698760986	6.1 m/s	209.4°
01.01.2015 03:00	2.03137302398682	5.8 m/s	203.1°

Figure 9.12: Screenshot - example of training step



Valsneset vindpark

General Information

STATUS
Ready to forecast

TRAINING STAGE
Training completed

TRAINING PROGRESS

URL NAME
valsneset-vindpark

ACCESS TOKEN
665a126c-827c-4e66-924f-8f4dbbf8a643

LATITUDE
63.819557

LONGITUDE
9.629051

MAX PRODUCTION
11.5 MW

DATE CREATED
07.06.2020

Visualization - All data

Data (first 1000)

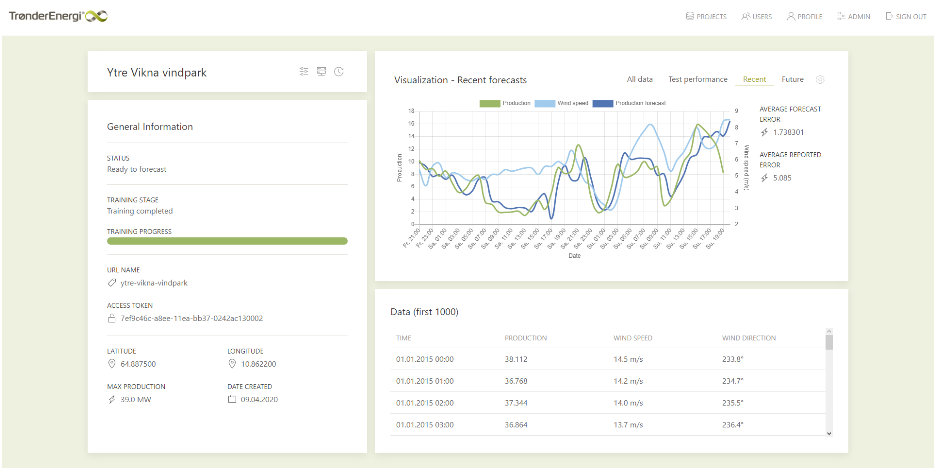
TIME	PRODUCTION	WIND SPEED	WIND DIRECTION
01.01.2015 00:00	2.36237478256226	6.7 m/s	220.4°
01.01.2015 01:00	2.86691999435425	6.4 m/s	215.2°
01.01.2015 02:00	2.35346698760986	6.1 m/s	209.4°
01.01.2015 03:00	2.03137302398682	5.8 m/s	203.1°

Figure 9.13: Screenshot - training completed, ready to forecast

Project page

This section will cover the project page, and specifically the features that it provides. Figure 9.14 shows the project page with park information, graph data and the first 1000 rows of the project data. The park name is located on the upper left side, just right of the park name there are three buttons. The first button is used to access the project settings as shown in figure Figure 9.15. The second button is a link to the Inference Service and the third button enables users to make forecasts using the Inference Service as shown in figure Figure 9.16.

The graph on the right side of the screen has four modes and a setting feature. The settings feature can be used to set the current resolution and date range of the graph, as long as it is not in recent/future forecasts mode. Figure 9.17 shows the view when the settings button is pressed. Figure 9.18 show how a specific data and resolution can be set, and Figure 9.19 shows the result. Figure 9.20, Figure 9.21, Figure 9.22 and Figure 9.23 show the four available graph modes.



Ytre Vikna vindpark

General Information

STATUS
Ready to forecast

TRAINING STAGE
Training completed

TRAINING PROGRESS

URL NAME
ytre-vikna-vindpark

ACCESS TOKEN
7ef9c46c-a8ee-11ea-bb37-0242ac130002

LATITUDE	LONGITUDE
64.887500	10.862200
MAX PRODUCTION	DATE CREATED
39.0 MW	09.04.2020

Visualization - Recent forecasts

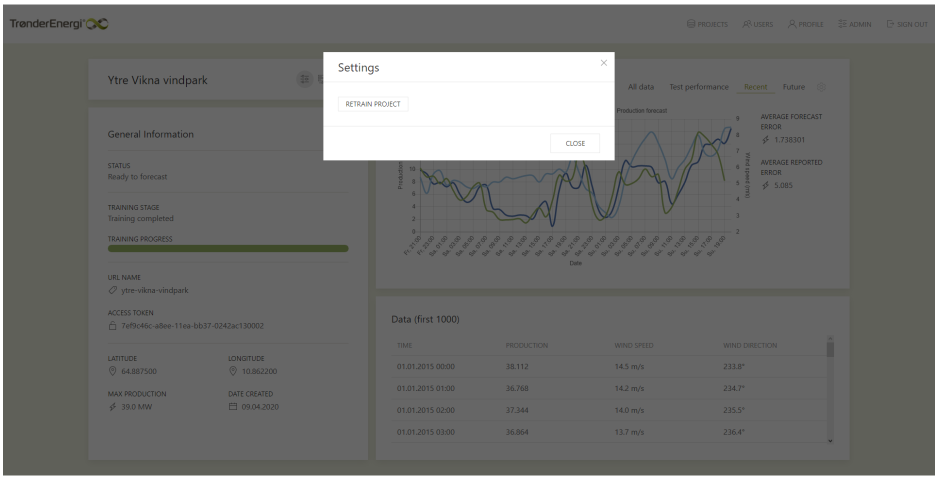
AVERAGE FORECAST ERROR
1.738301

AVERAGE REPORTED ERROR
5.085

Data (first 1000)

TIME	PRODUCTION	WIND SPEED	WIND DIRECTION
01.01.2015 00:00	38.112	14.5 m/s	233.8°
01.01.2015 01:00	36.768	14.2 m/s	234.7°
01.01.2015 02:00	37.344	14.0 m/s	235.5°
01.01.2015 03:00	36.864	13.7 m/s	236.4°

Figure 9.14: Screenshot - project page



Settings

Ytre Vikna vindpark

General Information

STATUS
Ready to forecast

TRAINING STAGE
Training completed

TRAINING PROGRESS

URL NAME
ytre-vikna-vindpark

ACCESS TOKEN
7ef9c46c-a8ee-11ea-bb37-0242ac130002

LATITUDE	LONGITUDE
64.887500	10.862200
MAX PRODUCTION	**DATE CREATED**
39.0 MW	09.04.2020

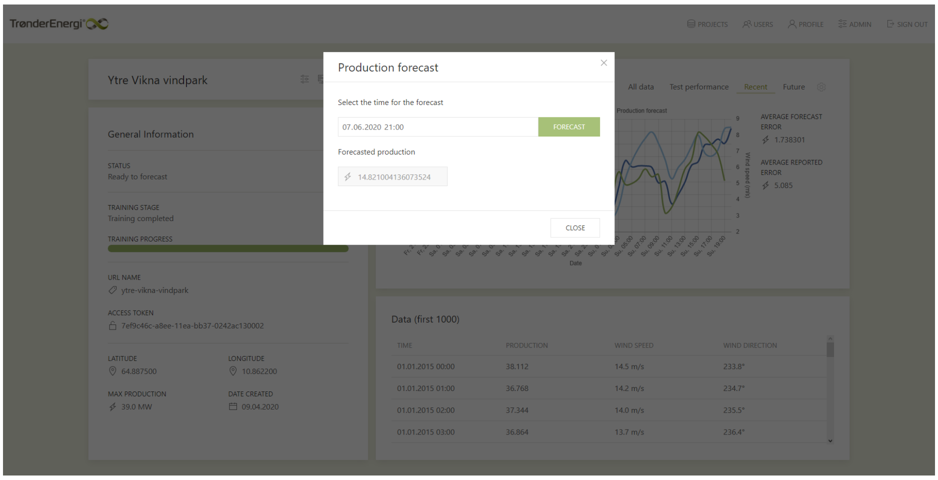
AVERAGE FORECAST ERROR
1.738301

AVERAGE REPORTED ERROR
5.085

Data (first 1000)

TIME	PRODUCTION	WIND SPEED	WIND DIRECTION
01.01.2015 00:00	38.112	14.5 m/s	233.8°
01.01.2015 01:00	36.768	14.2 m/s	234.7°
01.01.2015 02:00	37.344	14.0 m/s	235.5°
01.01.2015 03:00	36.864	13.7 m/s	236.4°

Figure 9.15: Screenshot - project settings



Production forecast

Select the time for the forecast
07.06.2020 21:00

Forecasted production
14.821004136073524

Ytre Vikna vindpark

General Information

STATUS
Ready to forecast

TRAINING STAGE
Training completed

TRAINING PROGRESS

URL NAME
ytre-vikna-vindpark

ACCESS TOKEN
7ef9c46c-a8ee-11ea-bb37-0242ac130002

LATITUDE	LONGITUDE
64.887500	10.862200
MAX PRODUCTION	**DATE CREATED**
39.0 MW	09.04.2020

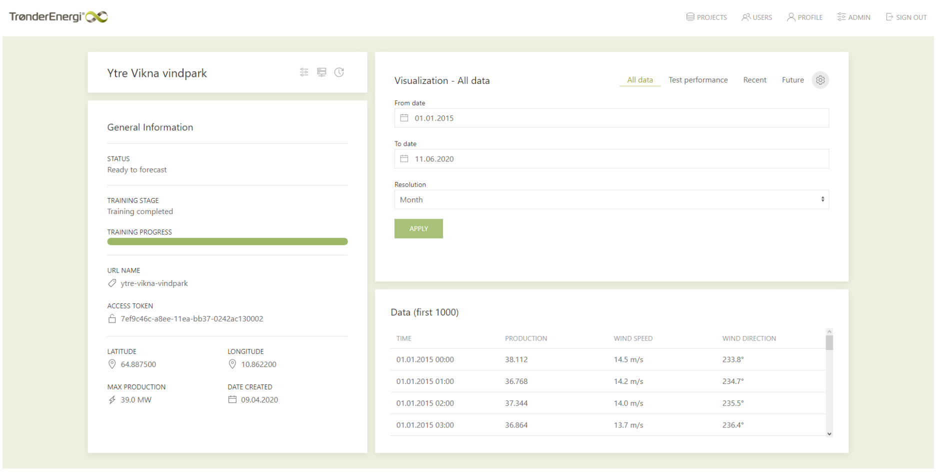
AVERAGE FORECAST ERROR
1.738301

AVERAGE REPORTED ERROR
5.085

Data (first 1000)

TIME	PRODUCTION	WIND SPEED	WIND DIRECTION
01.01.2015 00:00	38.112	14.5 m/s	233.8°
01.01.2015 01:00	36.768	14.2 m/s	234.7°
01.01.2015 02:00	37.344	14.0 m/s	235.5°
01.01.2015 03:00	36.864	13.7 m/s	236.4°

Figure 9.16: Screenshot - make forecast



Ytre Vikna vindpark

General Information

STATUS
Ready to forecast

TRAINING STAGE
Training completed

TRAINING PROGRESS

URL NAME
ytre-vikna-vindpark

ACCESS TOKEN
7ef9c46c-a8ee-11ea-bb37-0242ac130002

LATITUDE 64.887500	LONGITUDE 10.862200
MAX PRODUCTION 39.0 MW	DATE CREATED 09.04.2020

Visualization - All data

From date
01.01.2015

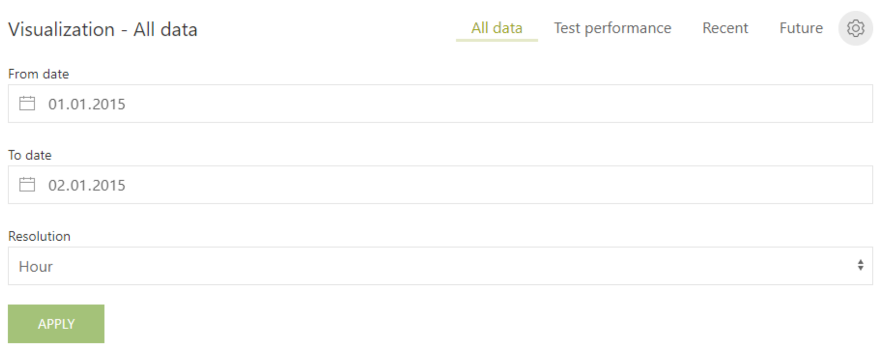
To date
11.06.2020

Resolution
Month

Data (first 1000)

TIME	PRODUCTION	WIND SPEED	WIND DIRECTION
01.01.2015 00:00	38.112	14.5 m/s	233.8°
01.01.2015 01:00	36.768	14.2 m/s	234.7°
01.01.2015 02:00	37.344	14.0 m/s	235.5°
01.01.2015 03:00	36.864	13.7 m/s	236.4°

Figure 9.17: Screenshot - graph settings



Visualization - All data

From date
01.01.2015

To date
02.01.2015

Resolution
Hour

Figure 9.18: Screenshot - graph settings close

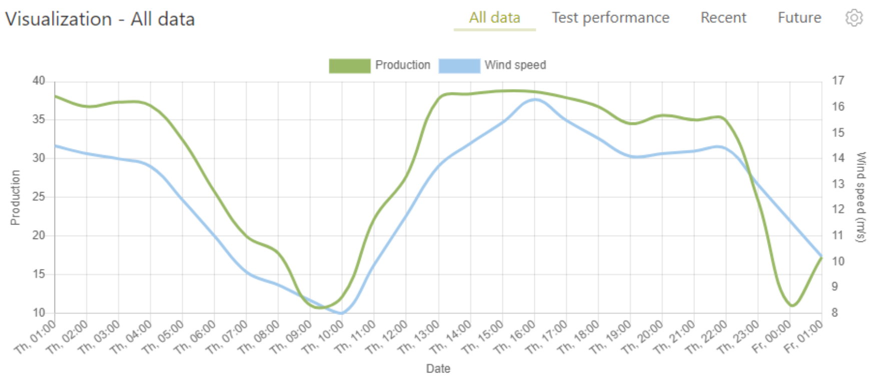


Figure 9.19: Screenshot - graph after setting date

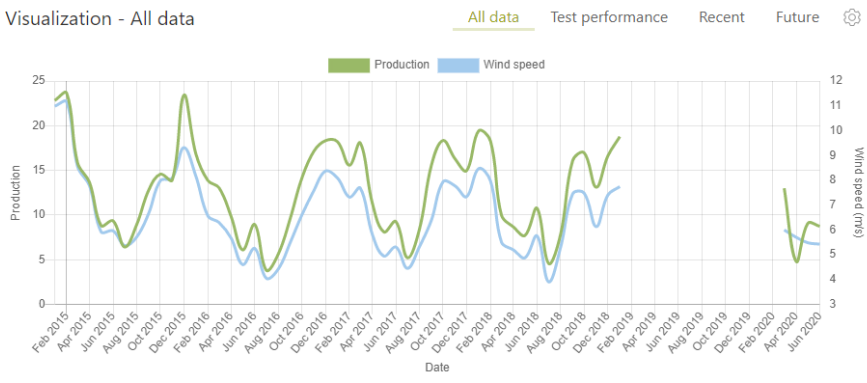
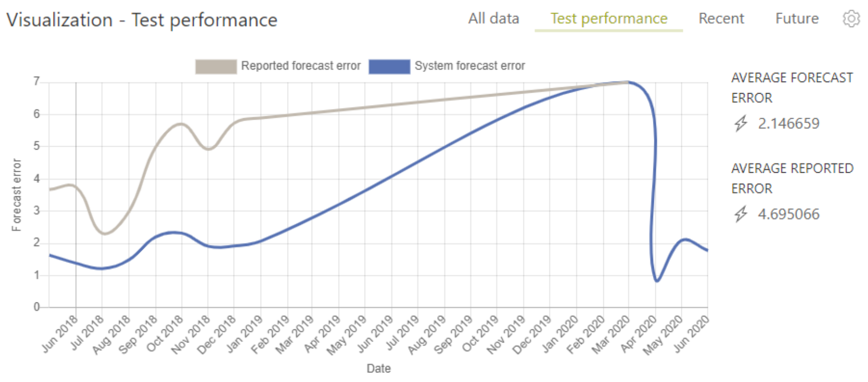


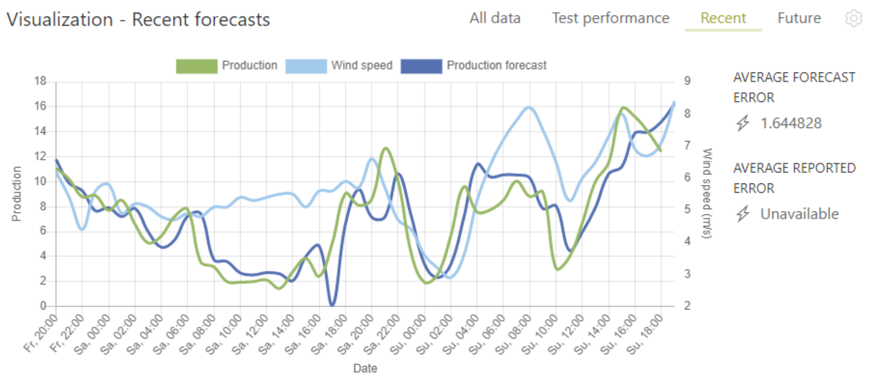
Figure 9.20: Screenshot - all data graph



AVERAGE FORECAST ERROR
2.146659

AVERAGE REPORTED ERROR
4.695066

Figure 9.21: Screenshot - test performance graph



AVERAGE FORECAST ERROR
1.644828

AVERAGE REPORTED ERROR
Unavailable

Figure 9.22: Screenshot - recent forecast graph

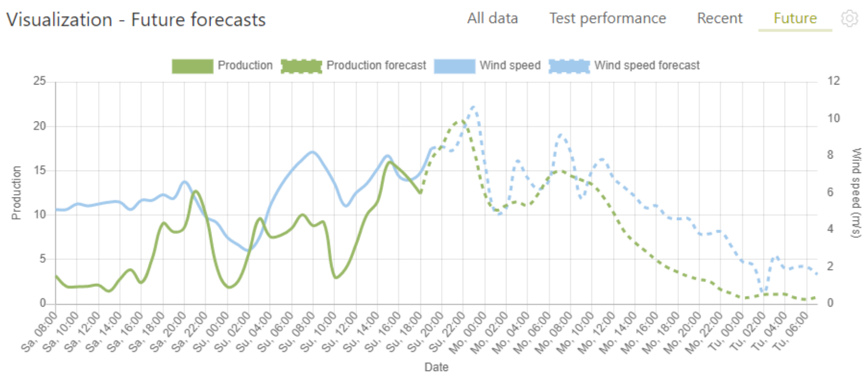
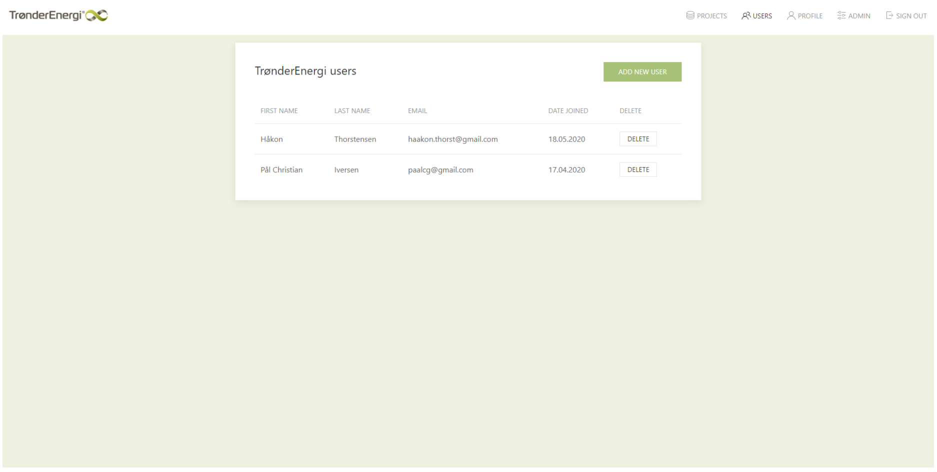


Figure 9.23: Screenshot - future forecast graph

User management

This section will introduce the user management tools available to authenticated users of the system. When pressing the user tab, the user will be presented with the screen shown in Figure 9.24. The screen lists all the users in the authenticated user's organization, with the option of viewing detailed information or deleting users. The detailed view can be seen in Figure 9.25. If the user press the delete button a modal will be shown, just like for projects. A authenticated user also have the capability to invite new users, in such a case the user will receive a welcome email with a link to set a new password. This form can be seen in Figure 9.26.

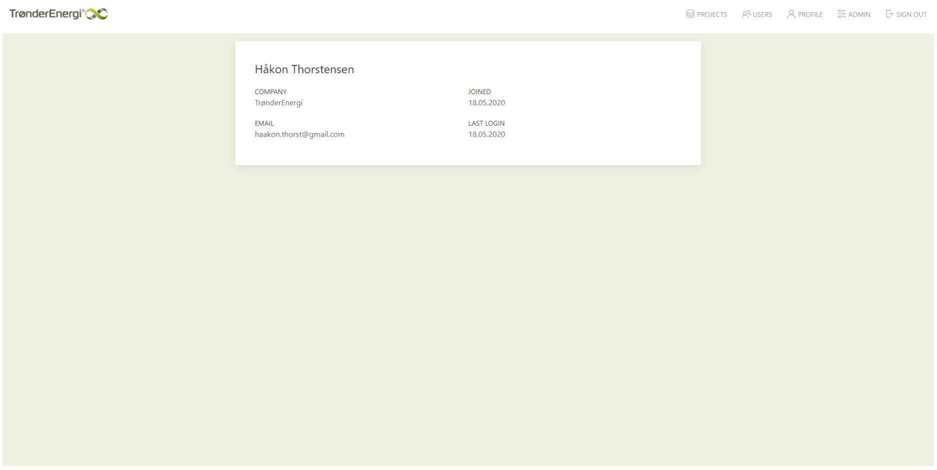
In addition to viewing details about other users, the authenticated user can also view his/hers own information. This functionality is seen in Figure 9.27. If the user choose to set a new password, the user will be shown the form in Figure 9.28.



TrønderEnergi users

FIRST NAME	LAST NAME	EMAIL	DATE JOINED	DELETE
Håkon	Thorstensen	haakon.thorst@gmail.com	18.05.2020	DELETE
Pål Christian	Iversen	paalcg@gmail.com	17.04.2020	DELETE

Figure 9.24: Screenshot - list of users



Håkon Thorstensen

COMPANY TrønderEnergi	JOINED 18.05.2020
EMAIL haakon.thorst@gmail.com	LAST LOGIN 18.05.2020

Figure 9.25: Screenshot - view detailed user information

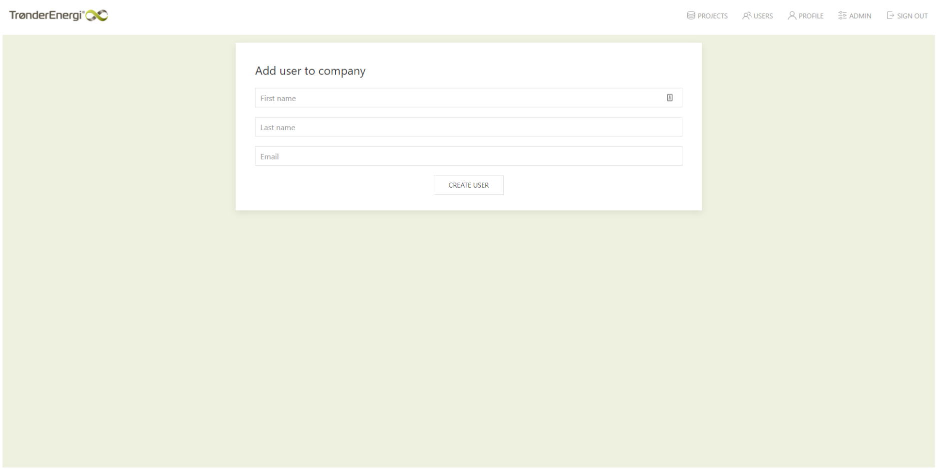
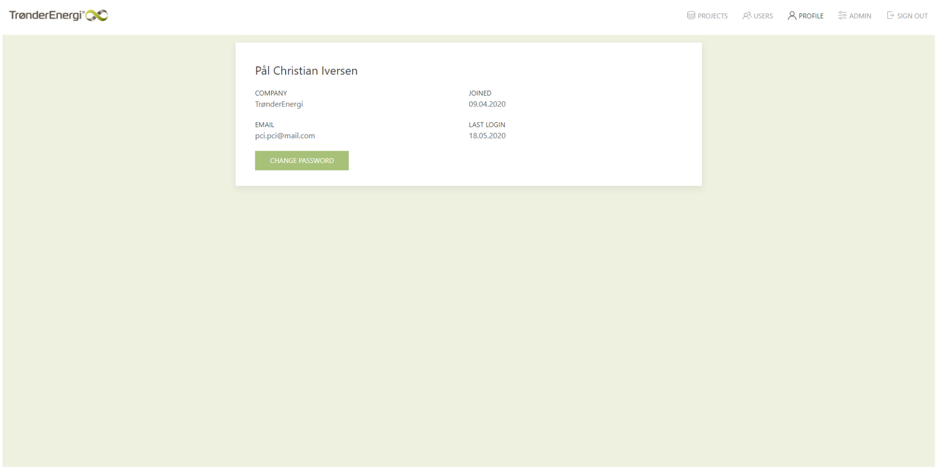


Figure 9.26: Screenshot - create new user



Pål Christian Iversen

COMPANY
TrønderEnergi

JOINED
09.04.2020

EMAIL
pci.pci@mail.com

LAST LOGIN
18.05.2020

Figure 9.27: Screenshot - profile for currently signed in user

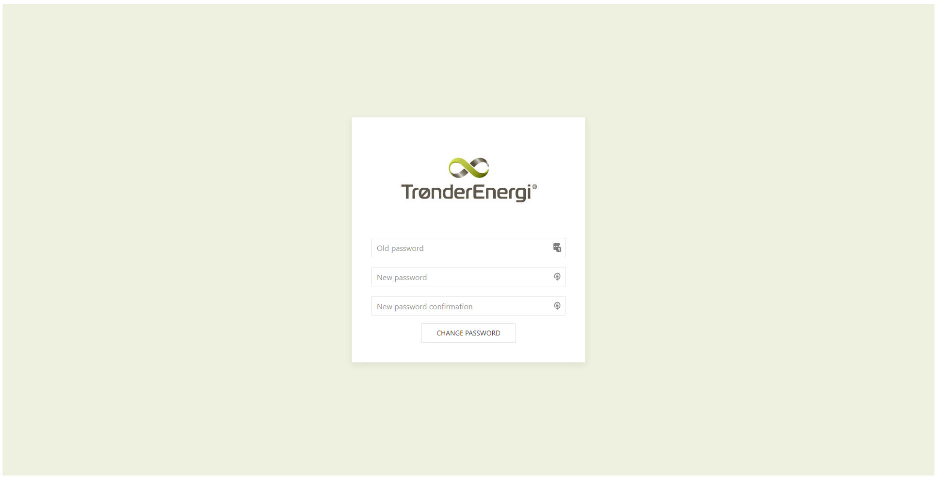
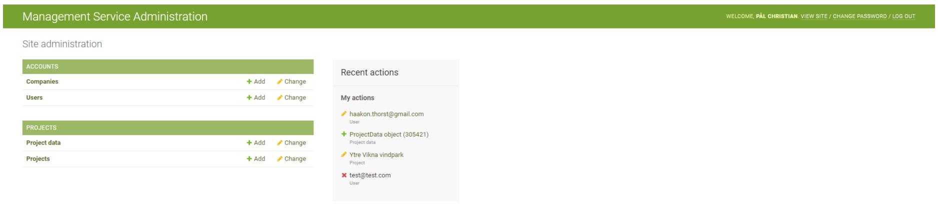


Figure 9.28: Screenshot - change password page

Administration

The administration site is only available to system administrators, and can be used to manage all aspects of the user and project functionality. Figure 9.29 shows the main administration page, it contains a list of the available models; users, companies, projects and project data. For each of them the administrator will be presented with a list as shown in Figure 9.30. If the administrator wish to edit a single entry it can be selected and the administrator will see a screen such as Figure 9.31.



Management Service Administration

WELCOME, PÅL CHRISTIAN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

ACCOUNTS		
Companies	Add	Change
Users	Add	Change

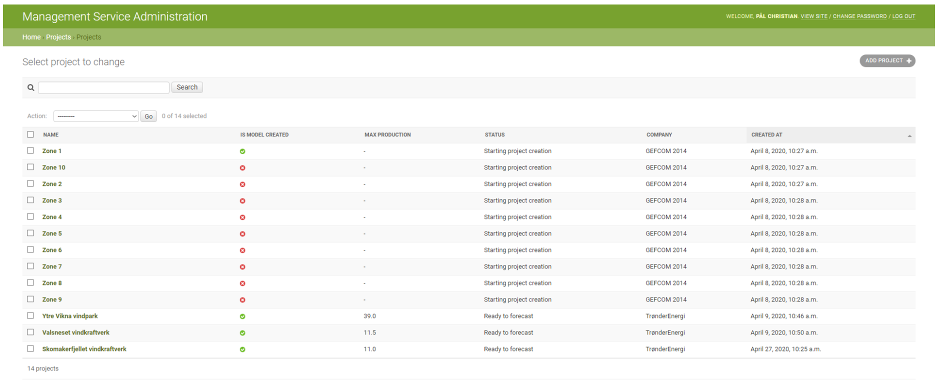
PROJECTS		
Project data	Add	Change
Projects	Add	Change

Recent actions

My actions

- haakon.thorst@gmail.com
  User
- ProjectData object (305421)
  Project data
- Ytre Vikna vindpark
  Project
- test@test.com
  User

Figure 9.29: Screenshot - overview of administration



Management Service Administration

WELCOME, PÅL CHRISTIAN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Projects › Projects

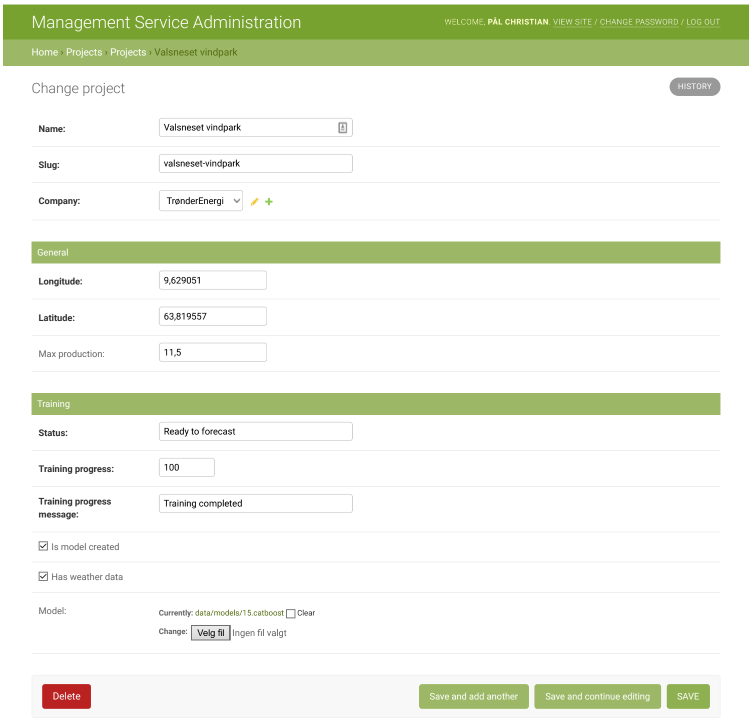
Select project to change

0 of 14 selected

NAME	IS MODEL CREATED	MAX PRODUCTION	STATUS	COMPANY	CREATED AT
Zone 1	✓	-	Starting project creation	GEFCOM 2014	April 8, 2020, 10:27 a.m.
Zone 10	✗	-	Starting project creation	GEFCOM 2014	April 8, 2020, 10:27 a.m.
Zone 2	✗	-	Starting project creation	GEFCOM 2014	April 8, 2020, 10:27 a.m.
Zone 3	✗	-	Starting project creation	GEFCOM 2014	April 8, 2020, 10:28 a.m.
Zone 4	✗	-	Starting project creation	GEFCOM 2014	April 8, 2020, 10:28 a.m.
Zone 5	✗	-	Starting project creation	GEFCOM 2014	April 8, 2020, 10:28 a.m.
Zone 6	✗	-	Starting project creation	GEFCOM 2014	April 8, 2020, 10:28 a.m.
Zone 7	✗	-	Starting project creation	GEFCOM 2014	April 8, 2020, 10:28 a.m.
Zone 8	✗	-	Starting project creation	GEFCOM 2014	April 8, 2020, 10:28 a.m.
Zone 9	✗	-	Starting project creation	GEFCOM 2014	April 8, 2020, 10:28 a.m.
Ytre Vikna vindpark	✓	39.0	Ready to forecast	TrønderEnergi	April 9, 2020, 10:46 a.m.
Valsneset vindkraftverk	✓	11.5	Ready to forecast	TrønderEnergi	April 9, 2020, 10:50 a.m.
Skomakerfjellet vindkraftverk	✓	11.0	Ready to forecast	TrønderEnergi	April 27, 2020, 10:25 a.m.

14 projects

Figure 9.30: Screenshot - admin - List of projects



Management Service Administration

WELCOME, PÅL CHRISTIAN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Projects › Projects › Valsneset vindpark

Change project

HISTORY

Name:	Valsneset vindpark
Slug:	valsneset-vindpark
Company:	TrønderEnergi

General

Longitude:	9,629051
Latitude:	63,819557
Max production:	11,5

Training

Status:	Ready to forecast
Training progress:	100
Training progress message:	Training completed

☑ Is model created

☑ Has weather data

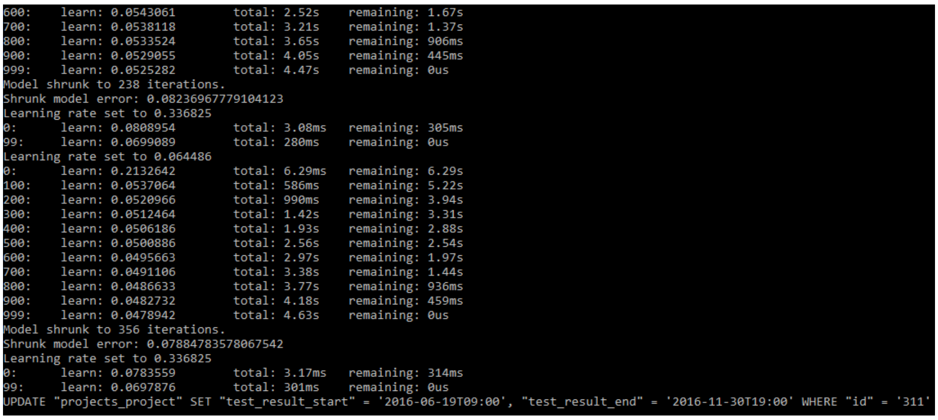
Model:	Currently: data/models/15.catboost ☐ Clear Change: Velg fil Ingen fil valgt

Delete · Save and add another · Save and continue editing · SAVE

Figure 9.31: Screenshot - admin - Change project

A.2 Training Service

The Training Service does not have a GUI, however Figure 9.32 shows the final command line output of a single training.

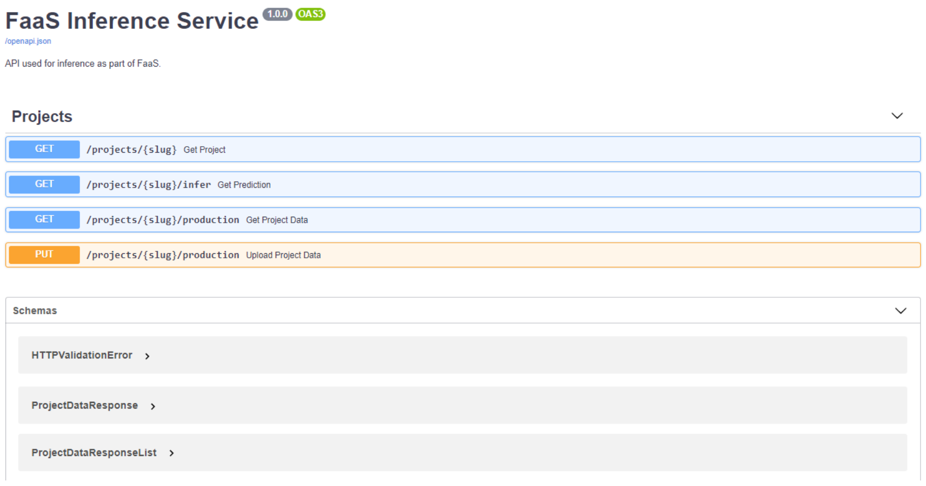


```
600:    learn: 0.0543061        total: 2.52s    remaining: 1.67s
700:    learn: 0.0538118        total: 3.21s    remaining: 1.37s
800:    learn: 0.0533524        total: 3.65s    remaining: 906ms
900:    learn: 0.0529055        total: 4.05s    remaining: 445ms
999:    learn: 0.0525282        total: 4.47s    remaining: 0us
Model shrunk to 238 iterations.
Shrunk model error: 0.08236967779104123
Learning rate set to 0.336825
0:      learn: 0.0808954        total: 3.08ms   remaining: 305ms
99:     learn: 0.0699089        total: 280ms    remaining: 0us
Learning rate set to 0.064486
0:      learn: 0.2132642        total: 6.29ms   remaining: 6.29s
100:    learn: 0.0537064        total: 586ms    remaining: 5.22s
200:    learn: 0.0520966        total: 990ms    remaining: 3.94s
300:    learn: 0.0512464        total: 1.42s    remaining: 3.31s
400:    learn: 0.0506186        total: 1.93s    remaining: 2.88s
500:    learn: 0.0500886        total: 2.56s    remaining: 2.54s
600:    learn: 0.0495663        total: 2.97s    remaining: 1.97s
700:    learn: 0.0491106        total: 3.38s    remaining: 1.44s
800:    learn: 0.0486633        total: 3.77s    remaining: 936ms
900:    learn: 0.0482732        total: 4.18s    remaining: 459ms
999:    learn: 0.0478942        total: 4.63s    remaining: 0us
Model shrunk to 356 iterations.
Shrunk model error: 0.07884783578067542
Learning rate set to 0.336825
0:      learn: 0.0783559        total: 3.17ms   remaining: 314ms
99:     learn: 0.0697876        total: 301ms    remaining: 0us
UPDATE "projects_project" SET "test_result_start" = '2016-06-19T09:00', "test_result_end" = '2016-11-30T19:00' WHERE "id" = '311'
```

Figure 9.32: Screenshot - Training result

A.3 Inference Service

The Inference Service comes with a list of available endpoints, as seen in Figure 9.33. Figure 9.34 shows an example of how the web interface can be used to access the endpoint and Figure 9.35 shows how the same data was accessed using the URL in a standard internet browser. As the Inference Service is primary intended to be accessed by systems, the figure are only examples of how it can be used.



FaaS Inference Service 1.0.0 OAS3

/openapi.json

API used for inference as part of FaaS.

Projects

GET /projects/{slug} Get Project

GET /projects/{slug}/infer Get Prediction

GET /projects/{slug}/production Get Project Data

PUT /projects/{slug}/production Upload Project Data

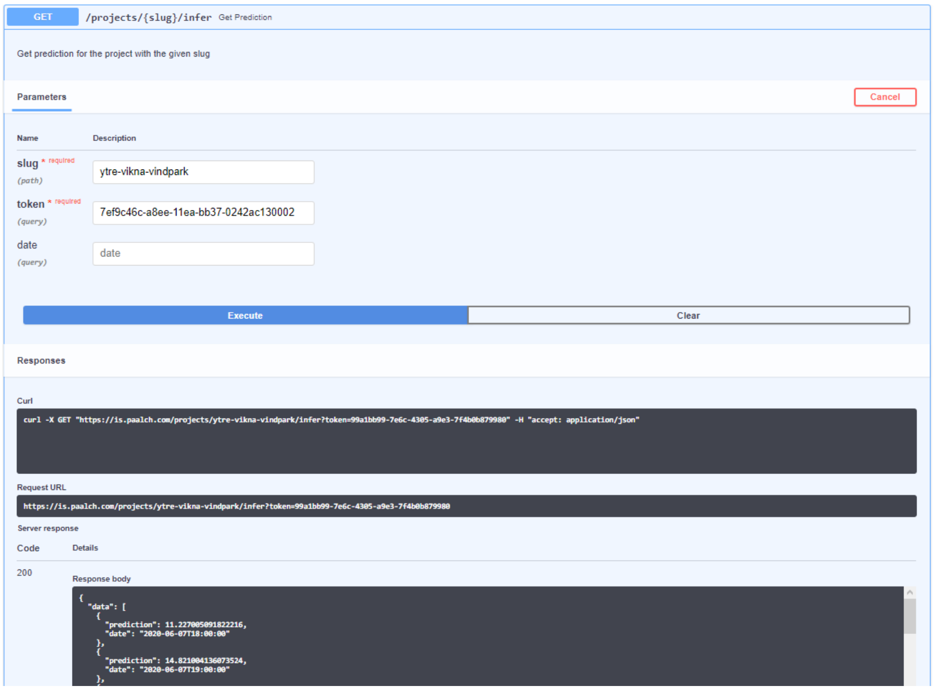
Schemas

HTTPValidationError

ProjectDataResponse

ProjectDataResponseList

Figure 9.33: Screenshot - Inference Service main page



GET /projects/{slug}/infer Get Prediction

Get prediction for the project with the given slug

Parameters

Cancel

Name Description

slug * required (path) ytre-vikna-vindpark

token * required (query) 7ef9c46c-a8ee-11ea-bb37-0242ac130002

date (query) date

Execute Clear

Responses

Curl

```
curl -X GET "https://is.paalch.com/projects/ytre-vikna-vindpark/infer?token=99a1bb99-7e6c-4305-a9e3-7f4b0b879980" -H "accept: application/json"
```

Request URL

```
https://is.paalch.com/projects/ytre-vikna-vindpark/infer?token=99a1bb99-7e6c-4305-a9e3-7f4b0b879980
```

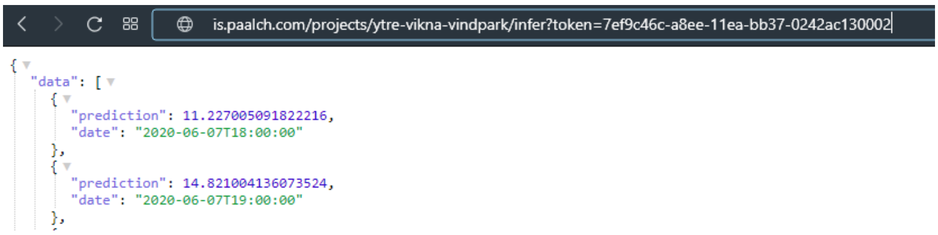
Server response

Code Details

200 Response body

```
{
  "data": [
    {
      "prediction": 11.227005091822216,
      "date": "2020-06-07T18:00:00"
    },
    {
      "prediction": 14.821004136073524,
      "date": "2020-06-07T19:00:00"
    },
```

Figure 9.34: Screenshot - project data using web interface



is.paalch.com/projects/ytre-vikna-vindpark/infer?token=7ef9c46c-a8ee-11ea-bb37-0242ac130002

```
{
  "data": [
    {
      "prediction": 11.227005091822216,
      "date": "2020-06-07T18:00:00"
    },
    {
      "prediction": 14.821004136073524,
      "date": "2020-06-07T19:00:00"
    },
```

Figure 9.35: Screenshot - project data using URL

P.C. Iversen, H. Thorstensen

Automatic Wind Power Forecasting as a Service

NTNU
Norwegian University of
Science and Technology