

Master Thesis



Czech
Technical
University
in Prague

F2

Faculty of Mechanical Engineering
Department of Instrumentation and Control Engineering

Gradient Boosting for Process Modeling in Smart Building Systems

Bc. Adéla Čekalová

Supervisor: Ing. Cyril Oswald, Ph.D.
January 2023



MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Čekalová Adéla** Personal ID number: **466529**
Faculty / Institute: **Faculty of Mechanical Engineering**
Department / Institute: **Department of Instrumentation and Control Engineering**
Study program: **Automation and Instrumentation Engineering**
Specialisation: **Automation and Industrial Informatics**

II. Master's thesis details

Master's thesis title in English:

Gradient Boosting for process modeling in smart building systems

Master's thesis title in Czech:

Gradient Boosting pro modelování procesů v systémech chytrých budov

Guidelines:

Study the use of Gradient Boosting for modeling processes in smart building systems and apply the selected approach to data supplied by Energocentrum plus s.r.o.

- Research the use of gradient boosting for modeling processes in smart building systems
- Research suitable libraries in Python programming language for application of the gradient boosting method on data supplied by a company Energocentrum plus s.r.o.
- Implement the gradient boosting method in Python for modeling of process systems in smart building systems
- Validate your implementation on data supplied by Energocentrum plus s.r.o.

Bibliography / sources:

Samir Touzani, Jessica Granderson, Samuel Fernandes, Gradient boosting machine for modeling the energy consumption of commercial buildings, Energy and Buildings, Volume 158, 2018, Pages 1533-1543, ISSN 0378-7788, <https://doi.org/10.1016/j.enbuild.2017.11.039>.
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>
Robert E. Schapire, The Boosting Approach to Machine Learning An Overview. MSRI Workshop on Nonlinear Estimation and Classification, 2002. <http://courses.csail.mit.edu/6.034f/ai3/msri.pdf>

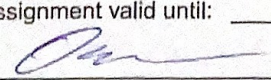
Name and workplace of master's thesis supervisor:

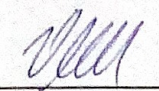
Ing. Cyril Oswald, Ph.D. U12110.3

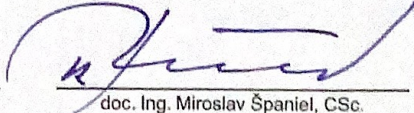
Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **27.10.2022** Deadline for master's thesis submission: **26.01.2023**

Assignment valid until: _____


Ing. Cyril Oswald, Ph.D.
Supervisor's signature


Head of department's signature


doc. Ing. Miroslav Španiel, CSc.
Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

7.11.2022
Date of assignment receipt


Student's signature

Acknowledgements

I pay my deep sense of gratitude to my advisor, Ing. Cyril Oswald, Ph.D., for his willingness, long-term support, and for guiding me through this whole master thesis.

Then, I would like to thank the company Energocentrum PLUS, s.r.o., for providing me with the data necessary for this work.

Last but not least, I express my sincere thanks to my family and friends, especially to my parents, for their patience, love, advice, and encouragement through my studies, I truly appreciate it.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague, January 19, 2023

Abstract

The main focus of this master thesis is the field of boosting algorithms in the domain of process modeling in smart building systems. The work deals with the creation and implementation of a machine learning model in Python, which learns to predict the input power consumption of buildings from the knowledge of the outdoor temperature, building occupancy, and timestamp.

Keywords: gradient boosting, machine learning, ensemble algorithms, decision trees, smart buildings, energy consumption, Python

Supervisor: Ing. Cyril Oswald, Ph.D.
Technická 1902/4,
Praha 6

Abstrakt

Náplní této diplomové práce je problematika boosting algoritmů v oblasti modelování procesů v systémech chytrých budov. Práce se zabývá vytvořením machine learning modelu v Pythonu, který se naučí predikovat příkon budov ze znalosti venkovní teploty, obsazenosti budovy, datumu a času.

Klíčová slova: gradient boosting, machine learning, ensemble algoritmy, rozhodovací stromy, chytré budovy, spotřeba energie, Python

Překlad názvu: Gradient Boosting pro modelování procesů v systémech chytrých budov

Contents

List of Abbreviations	1	5 Results	65
List of Symbols	2	5.1 Comparison of the Scikit-learn and the XGBoost Models	66
Introduction	3	5.2 Comparison of the XGBoost and the TOWT Models.....	69
1 Motivation and Goals	5	Conclusion	77
2 Theory	7	Bibliography	79
2.1 Machine Learning.....	7		
2.1.1 Machine Learning Challenges	11		
2.2 Decision Trees	13		
2.2.1 Stratification of the Feature Space	15		
2.2.2 Tree Pruning	16		
2.2.3 Tree Versus Linear Models ..	17		
2.2.4 Advantages and Disadvantages of Decision Trees	17		
2.3 Ensemble Methods	19		
2.3.1 Bagging	21		
2.3.2 Boosting	24		
3 Gradient Boosting Machine	29		
3.1 Methodology - Function Estimation	30		
3.1.1 Numerical Optimization	31		
3.1.2 Optimization in Function Space	32		
3.2 Gradient Boosting Algorithm...	32		
3.3 GBM Design	33		
3.3.1 Loss Functions.....	33		
3.3.2 Weak Learners.....	35		
3.3.3 Regularization Techniques...	36		
3.4 GBM Interpretation.....	39		
4 Model Implementation	41		
4.1 Data Preprocessing	41		
4.1.1 Loading and Analyzing The Data	42		
4.1.2 Handling Missing Values	48		
4.1.3 Adding Features	48		
4.1.4 Transforming Categorical Features - Encoding.....	49		
4.1.5 Removing Outliers	50		
4.1.6 Scaling	52		
4.1.7 Treating Correlated Features	53		
4.1.8 Splitting The Data	55		
4.2 Model Development	56		
4.2.1 Scikit Learn Model.....	56		
4.2.2 XGBoost Model	57		

Figures

<p>2.1 AI vs. ML vs. DL [1]..... 8</p> <p>2.2 Variance and bias analogy [2]... 11</p> <p>2.3 Decision tree structure and terminology 13</p> <p>2.4 Decision tree non-overlapping regions [3] 14</p> <p>2.5 General ensemble methods [4] .. 20</p> <p>2.6 Simple bootstrapping example [5] 21</p> <p>2.7 Bagging [6] 23</p> <p>2.8 Difference between bagging and boosting [7] 24</p> <p>2.9 Boosting - increasing weights of misclassified data points [8] 25</p> <p>2.10 AdaBoost Algorithm [8] 26</p> <p>3.1 Loss functions for continuous target variable y [9] 34</p> <p>3.2 Decision trees as weak/base learners [9] 36</p> <p>3.3 GBM regression overfitting example [9] 37</p> <p>3.4 Fitting a decision-tree GBM to a noisy data [9] 37</p> <p>3.5 Training set error vs. Validation set error [9] 38</p> <p>4.1 Dataset No. 29 44</p> <p>4.2 Dataset No. 33 46</p> <p>4.3 Dataset No. 63 46</p> <p>4.4 Occupancy pie chart for building No. 7 46</p> <p>4.5 Incorrect values of temperature for building No. 14..... 47</p> <p>4.6 Example of the dataset with outliers (building No. 22) 47</p> <p>4.7 Box plot for building No. 22 before removing outliers 51</p> <p>4.8 Box plot for building No. 22 after removing outliers 51</p> <p>4.9 Dataset No. 22 after outlier removal..... 51</p> <p>4.10 Histogram for building No. 22 before removing outliers 52</p> <p>4.11 Histogram for building No. 22 after removing outliers 52</p> <p>4.12 Heatmap for building No. 22 before dropping correlated fetures . 53</p>	<p>4.13 Heatmap for building No. 22 after dropping correlated fetures 54</p> <p>4.14 Pair plot for building No. 22 .. 55</p> <p>4.15 Feature importance plot for building No. 22..... 63</p> <p>4.16 The first decision tree in the ensemble for building No. 0 64</p> <p>5.1 R^2 scores histogram - Comparison of the SKLEARN model against the XGB model 66</p> <p>5.2 MSE histogram - Comparison of the SKLEARN model against the XGB model 67</p> <p>5.3 The biggest differences in R^2 scores of the SKLEARN and XGB models 67</p> <p>5.4 The biggest dissimilarities in MSE of the SKLEARN and XGB models 68</p> <p>5.5 Comparison of the true mean values against the mean values of TOWT and XGB models 69</p> <p>5.6 Comparison of the actual STD against the STD values of TOWT and XGB models 70</p> <p>5.7 Boxplots of the actual, XGB, and TOWT mean values..... 70</p> <p>5.8 Boxplots of the actual, XGB, and TOWT STD valuess 70</p> <p>5.9 R^2 scores histogram - Comparison of the TOWT model against the XGB model 71</p> <p>5.10 MSE histogram - Comparison of the TOWT model against the XGB model 71</p> <p>5.11 CV(RMSE) histogram - Comparison of the TOWT model against the XGB model..... 72</p> <p>5.12 The biggest differences in R^2 scores between the TOWT and XGB models 72</p> <p>5.13 The biggest dissimilarities in MSE values between the TOWT and XGB models 73</p> <p>5.14 Predictions of input power for building No. 108..... 74</p> <p>5.15 First week of predictions for building No. 108..... 74</p>
--	--

5.16 First week of predictions for building No. 29	74
--	----

Tables

2.1 ML Grouping	10
4.1 Example - Data for building number 29	43
4.2 One-hot encoding	49
4.3 Descriptive statistics before and after outliers removal using IQR ..	50
4.4 Performance improvement when public holidays incorporated	60
4.5 Performance improvement when altering loss functions	62
4.6 Final XGBoost Model's parameters	62
5.1 Evaluation of the overall models' performance	68
5.2 Evaluation of the overall models' performance	75



■ List of Abbreviations

AI	Artificial Intelligence
CatBoost	Category Boosting
CV(RMSE)	Coefficient of Variation of the Root Mean Squared Error
DL	Deep Learning
GBM	Gradient Boosting Machine
HVAC	Heating, Ventilation, and Air Conditioning
IoT	Internet of Things
KNN	K-Nearest Neighbor
LightGBM	Light Gradient Boosting Machine
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MSE	Mean Squared Error
NN	Neural Networks
PCA	Principal Component Analysis
RF	Random Forest
RMSE	Root Mean Squared Error
RPA	Robotic Process Automation
SSR	Sum of Squared Residuals
SVM	Support Vector Machines
TOWT	Time of Week and Temperature Model
XGBoost	Extreme Gradient Boosting

■ List of Symbols

δ	[1]	Cutting-edge parameter (robustness of Huber loss)
η	[1]	Subsample fraction
λ	[1]	Learning rate/Shrikage parameter
Ψ	[1]	Loss function
d	[1]	Depth of the decision tree
M	[1]	Number of iterations



Introduction

The world is constantly shifting to be more digital; thus, the real and the virtual parts are merging together. Even companies are adopting the current consumers' digital trends. The constant need for new gadgets, software products, robots, RPA ¹, and data warehouses requires more energy resources. In general, people's demand for electricity is rising with all these digital advancements.

According to the forecast by International Energy Agency, global electric power generation will increase by 63% in the nearest 30 years [10]. This is mainly driven by the economic growth of developing countries around the world.

4 main tendencies occur based on [10]:


- *Optimizing energy consumption of buildings*
- *Decentralization of electricity production*
- *Using renewable sources of energy*
- *Mastering the newest technologies, for example, IoT ²*

As non-fossil energy generation progresses and becomes more popular, the energy industry shifts to a more decentralized system. Now even individuals and companies are able to supply their own energy using solar panels, wind generators, and so on.

The U.S. Department of Energy and UNEP estimates that buildings' heating and cooling systems account for nearly 18-24% of all energy usage and produce 40% of total carbon footprint[10]. However, buildings certified as "green" throw 34% less carbon dioxide, consume 25% less energy, 11% less water, and that is the driver of the smart buildings trend which aims to alleviate climate change and reduce energy consumption. This tendency yields economical savings, resource efficiency as well as it is more environmentally friendly since it reduces greenhouse gas emissions. [10]

¹RPA = Robot Process Automation

²IoT = Internet of Things



With the help of industrial IoT and all the smart sensors, meters, and devices that commercial smart buildings are nowadays equipped with, high-frequency interval data can be obtained. These data allow various processes to be monitored, analyzed, and optimized, for instance, heating, ventilation, and air conditioning (HVAC) optimization, and malfunctions detection. This digital infrastructure and increased availability of data from sensors have led to possibilities of using advanced machine learning models for energy consumption prediction and other tasks. [11] [12]

Lately, ensemble algorithms gained more attention in predicting energy usage. Specifically, GBM ³ is a powerful machine learning algorithm that is being implemented in many data-driven fields.

In this thesis, gradient boosting methods of energy consumption prediction are proposed.

³GBM = Gradient Boosting Machine

Chapter 1

Motivation and Goals

The motivation of this thesis is to broaden the knowledge of machine learning algorithms that are used for data prediction, specifically energy consumption. In general, the ML field is constantly developing and improving, especially in terms of time and computational efficiency of algorithms. However, this work focuses mostly on ensemble methods, namely on boosting algorithms and their usage in process modeling in smart buildings. Due to high requirements for energy efficiency, and cost savings, optimization systems are deployed in smart buildings. They are used to analyze and monitor the building's operation by setting the right parameters to ensure the desired temperature inside the building for the employees to work comfortably.

The goal of this master thesis is to study the use of Gradient Boosting for modeling processes in smart building systems and apply the selected approach to data supplied by Energocentrum plus s.r.o. The outcome should be a gradient boosting model in Python that is able to predict the power needed for optimal operation of a particular building based on the knowledge of outside temperature, building's occupancy, and timestamp. In order to achieve so, there are some steps to be completed to fulfill this master thesis's assignment:

1. Research the use of gradient boosting modeling processes in smart building systems
2. Research suitable libraries in Python programming language for application of the gradient boosting method on data supplied by a company Energocentrum plus s.r.o.
3. Implement the gradient boosting method in Python for modeling of process systems in smart building systems
4. Validate the implementation on data supplied by Energocentrum plus s.r.o.

This thesis is divided into 7 chapters, including the introduction and the conclusion. The work is organized as follows:

- *Brief introduction to smart building and machine learning*
- *Explanation of decision trees*

- *An overview of ensemble methods*
- *The basic principles of Gradient Boosting Machine, which requires understanding the following terms:*
 - *Weak/Strong learner*
 - *Ensemble boosting technique*
- *Data analysis and preprocessing*
- *Research of Python libraries suitable for boosting algorithms*
- *Gradient Boosting model implementation in Python*
- *Critical evaluation of achieved results*

In the theory section, a brief overview of the machine learning field is provided. Then, ensemble methods, their advantages, disadvantages, and differences over each other are described. The majority of the theory part deals with gradient boosting algorithm that uses decision trees as weak learners. In order to understand the whole process, essential key technical terms, such as the ensemble algorithm, boosting, and decision trees, are explained.

The practical part aims to develop and implement a gradient boosting machine learning model in Python. To make the model as precise in predicting as possible, various model's hyper-parameters have to be tuned. Before approaching the actual model creation, the data from smart buildings provided by Energocentrum plus s.r.o. are analyzed and visualized. In order to use the data for creating and training the model, several data preprocessing steps are performed. It is desired to get as precise and accurate predictions as possible; therefore, the performance of two boosting libraries in Python is compared and model parameters are optimized. The final model is then validated on the provided data and tested against the already existing TOWT model provided by the company Energocentrum plus s.r.o.

Chapter 2

Theory

Prediction models are in general divided into 3 main categories [13]:

- *Physics-based*
- *Statistical*
- *Machine learning-based*

The physics-based models are very complex and difficult to obtain, on the other hand, statistical models perform with low accuracy. The ML-based models are able to yield promising results and operate with complex models at the same time, they have become sort of a standard in predictive modeling.

2.1 Machine Learning

Machine learning (ML), which includes a deep learning field, is a subfield of artificial intelligence. This hierarchical relationship is illustrated in Fig. 2.1.

Artificial Intelligence (AI)

Artificial intelligence is a machine or application that simulates human behavior and is embedded with an intelligent capacity to autonomously execute based on predictions. AI has basically two elements [14]:

- *Predictions*
- *Execution*

Firstly, predictive modeling is used to deliver predictions upon which the AI system makes autonomous responses, in other words, executes necessary steps to fulfill the predictions. [14]

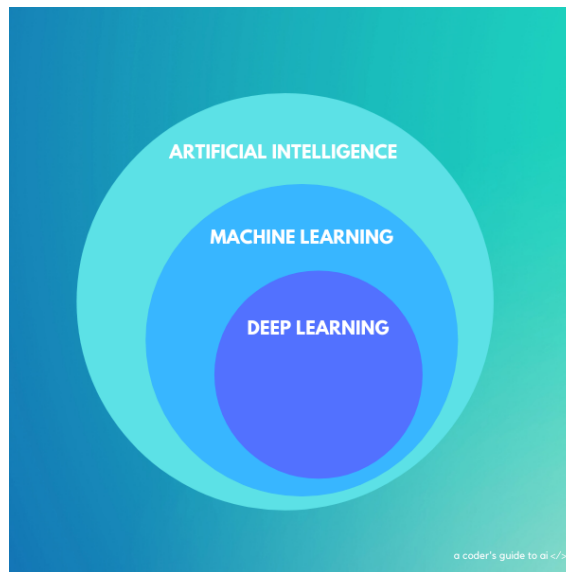


Figure 2.1: AI vs. ML vs. DL [1]

Deep Learning (DL)

Deep learning falls into the ML field and is established on NN¹. The core concept of the neural network is based on mathematical models of neurons which are supposed to mimic how actual neurons in the brain work. NNs are then trained on data and upon that, decisions are made. [1]

Machine Learning (ML)

Machine learning is the brain behind AI, it uses statistical methods to learn from data, find trends, and identify patterns in data. This information learned by the ML model is then used to make predictions about the future behavior of the data. The performance of ML algorithms gets better as it is exposed to more data.[14]

Since ML is a computer science discipline where algorithms learn from data that is provided to them, ML combines the following areas [14]:

- *Data mining and analysis*
- *Statistics*
- *Pattern recognition*
- *Computational learning theory in AI*
- *Mathematical optimization*

Some machine learning use cases [14]:

¹NN = Neural Network

- *Fraud /Spam detection*
- *Sales/Price/Consumption/Demand forecasting*
- *Natural Language Processing*
- *Assigning data to groups*
- *Recommendation engines*
- *Different computer vision tasks*
- ...

Types of machine learning methods [8] [15]:

- *Supervised Learning* = Training data contains both set of features and the known target (dependent) variable. This data is often referred to as labeled data.
- *Unsupervised Learning* = Predictions are made from unlabeled data, meaning that the set of input vectors has no target (dependent) variable. The goal of such a method is to determine the distribution of observed data, group similar data points, or decrease high-dimensional space.
- *Semi-Supervised Learning* = is a combination of the two methods above; thus, this technique uses both labeled and unlabeled data.
- *Reinforcement Learning* = is an ML area where an agent is supposed to find a suitable solution by taking action in an environment with the aim of maximizing the reward. As opposed to supervised learning, there are no labels, it is learning from experience and by trial and error.

An abundant amount of data is not everything because every method (learner) has some assumptions and knowledge behind its working principle; consequently, underlying knowledge of ML methods is crucial to be able to pick the right one for a certain application. So let's look at some ML algorithms. [1]

Types of ML algorithms by function [14]:

- *Classification*
- *Regression*
 - *Linear regression*
 - *Logistic regression*
- *Clustering*

- *Hierarchical methods*
- *K-means clustering*
- *Dimension reduction*
 - *Explanatory factor analysis*
 - *Principal component analysis (PCA)*
- *Deep learning*
- *Association rules*
- *Instance-based learning*
 - *K-nearest neighbor algorithm (KNN)*
- *Ensemble Methods*
 - *Random Forest*
 - *Gradient Boosting*
- *Decision Trees*
- *Bayesian statistics*
- *Regularization algorithms*

Table 2.1: ML Grouping

	Supervised Learning	Unsupervised Learning
Continuous	Regression	Dimensional reduction
Categorical	Classification	Clustering

ML is a very extensive area and finding the right algorithm to solve a specific task can be tremendously challenging to figure out. One of the main goals of designing and training an ML model is to generalize it well so that it shows good results not only for the data it is trained on but as well for new data. If it is not properly generalized, the model suffers from an overfitting issue. *Overfitting* means that the model fits the training data too perfectly and when new data points are presented to the model, its performance is significantly lower. Overfitting is overall a common undesirable problem in ML applications, there are several methods to prevent it, and some relevant ones will be mentioned later.

ML models for estimating energy consumption are often regression models using supervised learning techniques; therefore, the rest of the thesis is focused on this particular area of ML.

2.1.1 Machine Learning Challenges

Machine learning models combine data with knowledge, they cannot make up reasonable results from nothing.

Overfitting

As mentioned before, one of the main challenges in ML is overfitting. Overfitting occurs when a model is overly complex and usually has too many parameters in comparison to the number of observations in the training data. As a result, the model performs well on the training data, it learns the data including the noise perfectly, but it does not generalize well to new, unseen data points. [2]

Another way to interpret overfitting is by decomposing the error into bias and variance. [16]

Bias refers to the systematic errors that are introduced to the model during the training phase. As a consequence, the model has a tendency to learn the same incorrect things repeatedly. For example, if the training data contains a disproportionate number of examples from one group compared to another, the model may be biased towards classifying examples from the more common group more accurately. [2] [16]

Variance is the amount of deviation or spread in the predictions made by the model. When the variance is too high, it can lead to overfitting and too low variance can cause underfitting of the model. Considering a decision tree model, which is explained in Section 2.2, decision trees can suffer from high variance and potentially overfitting when they are too deep. In other words, they can be overly complex and sensitive to specific details of the training data. [2] [16]

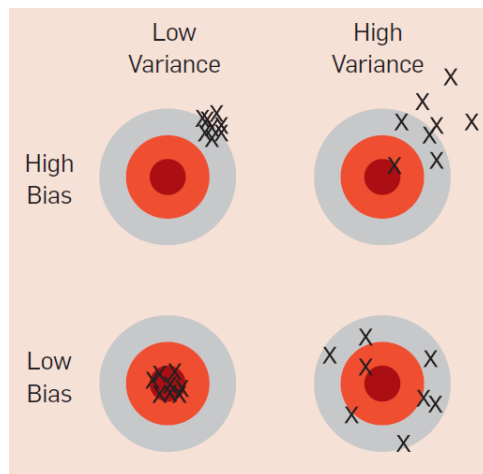


Figure 2.2: Variance and bias analogy [2]

The difference between bias and variance can be observed in a simple example of dart-throwing in Fig. 2.2.

Possible method to prevent overfitting [2]:

- *Gathering more data (if possible) or use a simpler model*
- *Adding regularization*

- *Using cross-validation*
- *Using early stopping*
- *Performing a statistical significance test, e.g. chi-square*

It is quite easy to get rid of overfitting (variance) by amplifying the opposite error of underfitting (bias). Avoiding both of these errors is not a simple task, as it would require training a perfect model. [2]

Curse of Dimensionality

Another big problem in the machine learning area is the curse of dimensionality. It arises when dealing with high-dimensional data, such as data with many features. Not only is such data difficult to analyze and visualize, but also the models perform poorly on it. The bad performance of the model is mainly caused by the fact that the number of data points required to accurately model a phenomenon increases exponentially with the number of dimensions. Meaning that if the model requires 100 data points to make accurate predictions in 2D, it would probably require about 10000 data points in 3D. Thus, adding more and more features is not always beneficial. [2]

Feature Engineering

What makes some of the machine learning models do very well and some of them fail? One of the most important components of creating an ML model is feature engineering. Features are inputs that the model is learning from and can significantly impact the performance of the model; hence, obtaining the right features is crucial to the good performance of the model. Feature engineering is the process of extracting useful features from the provided raw data, various statistical or machine learning methods can be used for it. [2]

The ideal state would be to have independent features that have a strong correlation with the class. Usually, the raw data is not in a form suitable for training. Then feature engineering comes into place with the target of deriving new features and adding them to the training set to improve the accuracy of the ML model. [17]

Feature engineering has multiple steps [17]:

- *Feature Creation*
- *Transformations - transforming one feature representation to another*
- *Feature Extraction - extracting useful information from raw data*
- *Exploratory Data Analysis - used in order to understand the data, create new hypotheses, and find patterns*
- *Benchmark model - running the developed model against benchmark model and comparing the resulting performance*

2.2 Decision Trees

Decision trees are a very popular algorithm that can be used for both classification and regression tasks. It uses an inverted tree-like structure to capture the relationship between independent variables (features) and the dependent variable (target), the structure is demonstrated in Fig. 2.3. Decision trees can make predictions from nonlinear data and can cope with both continuous and categorical data at the same time. This algorithm divides the input space into regions, where each region is associated with the corresponding node and is denoted by a different set of parameters. As a consequence, the space is divided into non-overlapping regions expressed in Fig. 2.4. [18] [19]

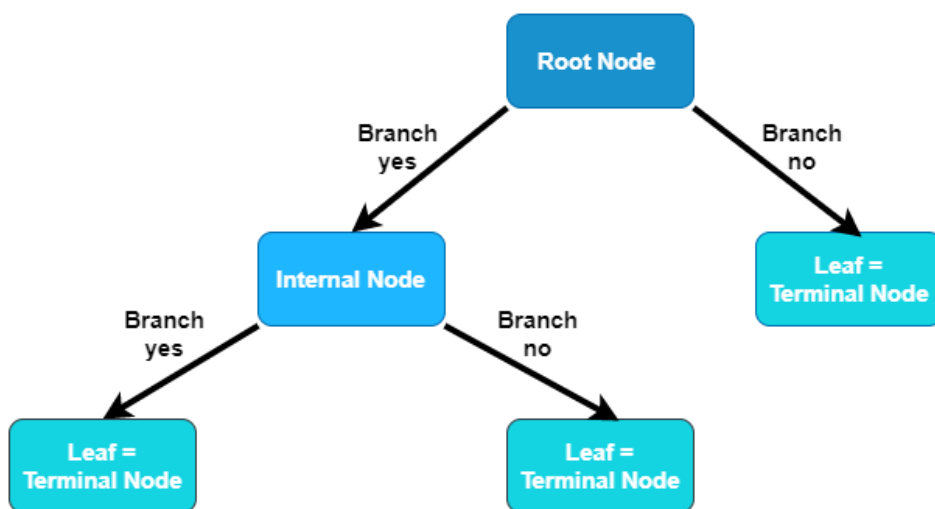


Figure 2.3: Decision tree structure and terminology

Decision's tree terminology [3]:

- *Leaves, also called terminal nodes*
 - represent the final regions
 - do not split any further
 - tree are displayed upside down → leaves are at the bottom of the tree
- *Internal (Decision) nodes* = nodes, where the decisions are made; hence, where the predictor space is split
- *Branches* = the connections between the nodes

Decision trees are quite transparent and easy to interpret, they can be expressed as a set of if-then-else rules. The topmost node, sometimes called

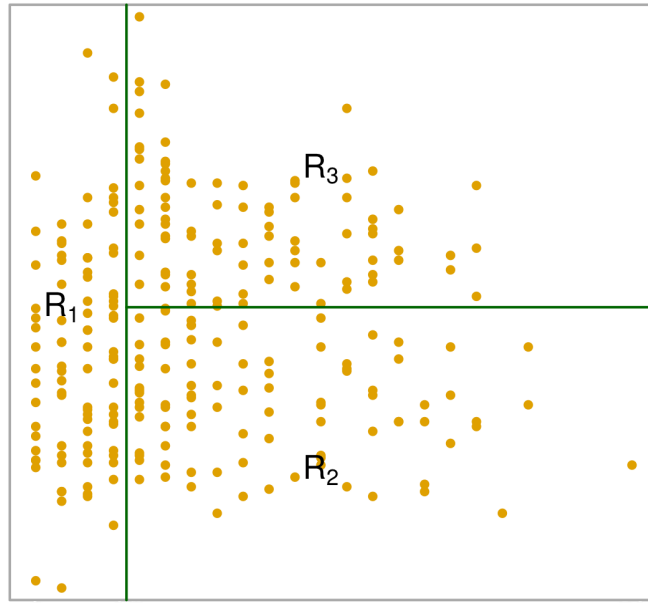


Figure 2.4: Decision tree non-overlapping regions [3]

the root node, contains the most important factor that plays a role in determining the target variable, then the algorithm moves step by step to less and less significant features. Each node focuses on one feature only. Since the regions are non-overlapping, one node always corresponds to one region. A decision tree algorithm is generally considered to be non-parametric unless the size of the tree is restricted to some maximum number, then it becomes parametric, which is usually the case. [3] [14]

There are two types of decision trees [14]:

- *Classification tree* = Categorical variable decision tree
 - Target variable is a categorical value
- *Regression tree* = Continuous variable decision tree
 - Target variable is a continuous value

A categorical variable is a variable that can take on any discrete value from a limited amount of values, usually some classes, for example, gender, ethnicity, color, type of car, etc.

On the other hand, a continuous variable is a variable that can gain any numerical value within a certain range, such as weight, temperature, and time.

Since this thesis deals with a continuous target variable, only **regression trees** will be discussed further.

2.2.1 Stratification of the Feature Space

There are two main steps in constructing a regression tree [3]:

1. Dividing the predictor space into J non-overlapping regions R_1, R_2, \dots, R_J .
2. Every observation X_1, X_2, \dots, X_P falls into one of the regions R_J . For each region R_J , the same predictions are made.

Taking into consideration regions R_1, R_2, R_3 in Fig. 2.4, the final response of the region is calculated as a mean of the training observations that are included in the specific region R_J . Let's assume that the mean response of the first region R_1 is 50, then for a given observation $X = x$, if $x \in R_1$, then the model predicts 50 as the target variable for such observation. [3] [20]

Recursive binary splitting is a method that is used to divide the feature space into regions. [18]

Recursive binary splitting is [3]:

- *Greedy* - Means that this approach does not look ahead, but it always makes the best split at the individual node.
- *Top-down* - The algorithm begins at the root node, where there is only one region that includes all the observations. Based on the rule (the most important feature) at the root node, the observations are split into two regions. This technique continues splitting the space into regions at each internal node until leaf nodes are reached, then the process stops and the final number of regions is obtained.

Splitting is performed in a way that minimizes the SSR². [18] First, the predictor X_J and the cutpoint s are chosen at the particular node such that the predictor space is divided into 2 subregions [3]: $R_1(j,s) = \{X|X_J < s\}$ and $R_2(j,s) = \{X|X_J \geq s\}$.

In simple words, it means that when the observation X evaluated based on the splitting feature of the node X_J takes on a value less than s , it goes to the region R_1 ; otherwise, it belongs to region R_2 . The aim is to find values j and s that minimize the equation [3]

$$\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2, \quad (2.1)$$

where \hat{y}_{R_1} is the mean response for the training observation in the first region $R_1(j,s)$ and \hat{y}_{R_2} is the mean response for the training observation in the other region $R_2(j,s)$. When the number of features p is not too big, the recursive binary splitting is completed quite fast. The value of SSR denotes the difference between the target variables of the training observations and

²SSR = Sum of Squared Residuals

their average (mean) value. The higher the SSR value, the higher variability (more dissimilarity) in the data. A lower value of SSR signifies lower variability. [3]

This process keeps repeating for all of the following nodes where the observations are divided into more and more regions based on a specific feature rule of each node. Recursive binary splitting continues until a stopping criterion is achieved. Consequently, leaf nodes are created and the algorithm calculates mean responses for each of the regions R_1, R_2, \dots, R_J based upon the training observations that ended up in the region. [3]

2.2.2 Tree Pruning

Splitting predictor space into non-overlapping regions described in 2.2.1 might yield promising results in the training phase, but is prone to overfit the training data. The reason is that the constructed tree might be too complex and not able to generalize well; as a result, it performs poorly on the test data. To overcome this problem, a method called *Pruning* is often used. The idea behind pruning is to remove some of the branches of the tree that have no significant contribution to making predictions. There are two general types of pruning [18]: *pre-pruning* and *post-pruning*. Pre-pruning techniques limit the size of the tree during the recursive partitioning process resulting in smaller trees. Smaller trees may have lower variance, but they are short-sighted and can omit some important patterns. On the contrary, post-pruning methods leave the whole decision tree to be trained and then remove some non-significant branches in order not to miss any notable patterns. Post-pruning is not as efficient as pre-pruning since it's more computationally expensive; however, it yields better results. The goal of post-pruning is to get a subtree that delivers the lowest error rate on the test multiset.[3]

Two popular tree pruning methods [18]:

- *Hold-out test*
 - simplest, fastest
 - going through each leaf node and evaluating whether its removal would improve the overall cost function of the test multiset -> the algorithm stops removing nodes when no more enhancement can be achieved
- *Cost-complexity pruning - also called weakest link pruning*

Let's briefly describe the cost-complexity pruning [3]:

1. Letting a large tree grow using the recursive binary splitting.
2. By applying cost complexity pruning on the large tree, a sequence of best subtrees is acquired. These trees are a function of parameter α .

3. Parameter α is derived using K-fold cross-validation. The training multiset is split up into K folds ($k = 1, \dots, K$). Steps 1 and 2 are done on all the folds except the k th one which is left out for evaluation of the error.
4. Algorithm gives back the subtree that corresponds to the α value that minimizes the error.

2.2.3 Tree Versus Linear Models

Both linear regression and regression trees are models that predict numerical variables. However, they differ quite a lot. It cannot be easily said which model is better as it depends on the relationship between the features and the response (target variable). If the relationship shows a linear trend, linear regression models might be a good fit for that. [3]

If the relationship between features and responses is complex and rather non-linear, regression trees can deal with that; and therefore, yield more accurate results than linear regression. [3]

Both linear regression and regression trees have different strengths and drawbacks. Linear regression is a simple and fast method that is easy to interpret, but it may not be able to capture more complex relationships. Regression trees, on the other hand, can capture more complex relationships, but they are prone to overfitting.

2.2.4 Advantages and Disadvantages of Decision Trees

Decision trees have plenty of advantages as well as limitations and drawbacks [3] [14]:

- ↑ *Non-linear relationships between variables can be modeled*
 - ↑ *Easy to be explained and interpreted*
 - ↑ *Easy to be displayed graphically which helps with interpretation*
 - ↑ *Can handle both categorical and numerical data at the same time*
 - ↑ *Can cope with categorical data without the need of encoding*
 - ↑ *Require less data → suitable for applications where data is sparse*
-
- ↓ *Non-robust = a small change in the data can significantly affect the predictions*
 - ↓ *Sensitive to training data - splitting rules depend on the training data*

- ↓ *Global optimum not guaranteed because of the greedy algorithm that does not look ahead*
- ↓ *In general, they do not reach very high accuracy as some other regression algorithms*
- ↓ *May not generalize well since they can feature high depth (overfitting)*

The performance of one single decision tree is not that good; however, combining multiple decision trees together brings considerable improvement. This concept of ensemble methods is introduced in the next Section 2.3.

2.3 Ensemble Methods

Machine learning methods have been developing and progressing for a couple of decades, including the field of ensemble methods, where some of the algorithms were proposed as early as the 1980s and 1990s.

In recent years, ensemble methods have gained significant attention and become more popular due to the success of methods such as random forests and gradient boosting machines in a variety of machine learning competitions and real-world applications. Overall, ensemble methods are used more and more since machine learning practitioners have come to recognize their ability to improve the performance, accuracy, and robustness of models. [11]

Ensemble methods are techniques that train multiple simple models, also called *base or weak learners*, and then aggregate them into one more powerful model (*strong learner*). Weak learners usually do not perform very well on their own; however, they are better than luck (coin flip); thus, combining them together improves the predictions gradually. Based on [2], it has been shown that combining more weak learners reach better results than trying to find one "best" learner; moreover, it doesn't cost much additional effort to do so. Even though ensemble machine learning algorithms have already been used successfully in many various areas, they have just quite recently been incorporated to solve tasks in the field of modeling energy processes of commercial buildings. [11] [2]

There are two techniques on how to build an ensemble, base learners can be generated either sequentially or in parallel. In parallel ensemble techniques, e.g. random forest (bagging), all the weak learners are trained at the same time and their predictions are combined to make the final prediction. This approach promotes the independence between weak learners. When it comes to sequential techniques, weak learners are trained one after the other, and the predictions of each base learner are then used to train the following base learner in the sequence. Therefore, the following learner is attempting to correct the mistakes of the base learner before, this makes the base learners dependent. An example of a sequential ensemble technique is AdaBoost (boosting method). [4]

The choice between the parallel and sequential technique depends on the specific application. Both methods should improve the accuracy and performance of the model. Sequential ensemble techniques are generally more computationally expensive, but they can achieve better accuracy in some cases. Parallel ensemble approaches are normally faster to train, but they may not be as accurate as sequential techniques in some cases.

Ensemble methods can be split into different subgroups based upon various aspects. Here are three main types of ensemble method [4]:

- *Bagging (=Bootstrap Aggregating)*: This involves training multiple models on different subsets of the training data and averaging their predictions.

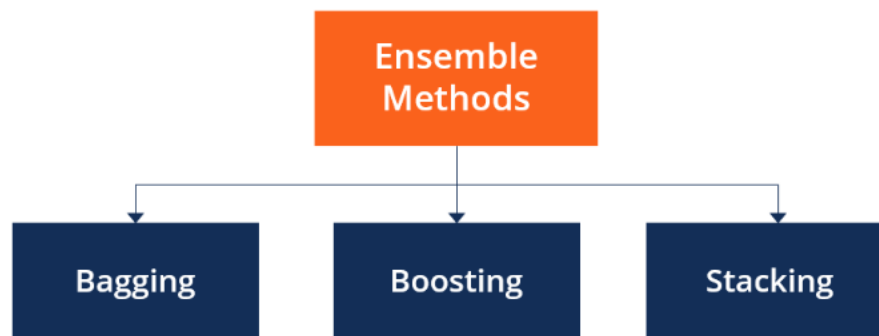


Figure 2.5: General ensemble methods [4]

Bagging can help reduce the variance of the model, as each model is trained on a different subset of the data.

- *Boosting*: In this case, models are trained sequentially, where each model tries to correct the mistakes of the previous model. The final prediction is made by combining the predictions of all the models. Boosting can help reduce bias and improve the performance of the model.
- *Stacking*: Again, multiple models are trained and their predictions (outputs) are used as features (inputs) for a higher-level model, which makes the final prediction.

Another differentiation of ensembles can be made depending on the base learners [16]:

- *Homogeneous Ensembles*
 - e.g. Random Forest, AdaBoost, XGBoost
 - use the same algorithm for each weak learner
 - many famous ensemble methods are of this type
- *Heterogeneous Ensembles*
 - weak learners are of different types
 - are built mostly independently
- *Metamodeling*
 - multiple models are working together to solve a problem
 - learners might be sending certain information to each other - essentially models receive as input the outputs of other models, e.g. stacking
 - example - one model can predict risk and the rest can classify the type of the risk (it's not a simple average as with bagging)

- this term is not as agreed upon as the two previous ones

Decision trees are in general susceptible to overfitting; hence, using an ensemble of decision trees typically improves the predicting accuracy by a lot.

Lets' explore bagging and boosting closely.

■ 2.3.1 Bagging

Bagging got its name from the fact that it combines bootstrapping and aggregation. The basic idea behind bagging is to train multiple base learners on the subsets of the train data and then let the learners vote on the output for the test dataset. Voting is performed by model averaging, it works well with the assumption that base learners make non-identical errors on the test set. [5]

When weak learners make independent errors from each other on the test set (non-overlapping error regions), model averaging works well. Considering k regression models with variance v and covariance c . Supposing that weak learners make exactly the same errors, then the errors are perfectly correlated and $c = v$. As a result, the mean squared error equals v and in that case, model averaging didn't help at all to elevate the performance. On the other hand, when errors of each weak learner are uncorrelated and $c = 0$, the mean squared error lowers down to $\frac{1}{k}v$, which would be the ideal case. In the worst case, the ensemble's performance equals to any of its components (learners) performance. If the learners make different errors, the ensemble's predictions are substantially better than the predictions of any of the individual base learners. [5]

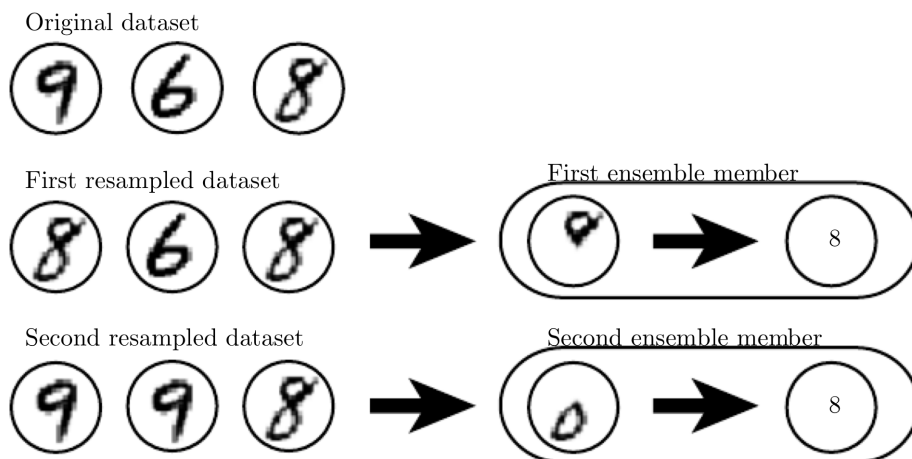


Figure 2.6: Simple bootstrapping example [5]

Bootstrapping

Bootstrapping is a resampling method that generates datasets of the same size as the original one by sampling with replacement from the initial population. In bagging, k different datasets are constructed using bootstrapping. Since sampling with replacement is employed, each new dataset most probably does not include all of the data points from the original dataset, some samples are missing and some are duplicated. Generally, $\frac{2}{3}$ of initial observations are found in the constructed dataset. By sampling the original dataset with replacement, bootstrapping ensures that each subset of the data is somewhat different from the rest, which helps to create diverse training datasets that are later used for training. [5]

A basic example of bootstrapping is demonstrated in Fig. 2.6, where the original dataset consists of numbers: 9, 6, 8. By bootstrapping, two distinct train datasets are created, in one of them, an 8 is repeated and in the other one, a 9 is duplicated. The goal is to be able to detect an 8. After training, the first learner is able to recognize the upper loop and the other one detects the bottom loop of an 8. Each of the models is fragile in making correct predictions, but when averaging their outputs, a more robust model is obtained. [5]

Aggregation

In the context of bagging, aggregation refers to the process of combining the predictions of the individual models trained on different subsets of the original data. It is performed either through simple averaging or by weighting the predictions according to the performance of the individual models. By aggregating the predictions of the base learners, bagging aims to improve the generalizability of the final model and reduce the risk of overfitting to the training data. [5]

Bagging, as captured in Fig. 2.7, can improve predictions for a lot of regression methods, but it is especially useful for decision trees. Considering decision trees as base learners, each decision tree is built in a slightly different manner because of the distinct train dataset and that ensures reducing the variance by averaging the results of all the weak learners. The final model is more general and has improved accuracy. [5] However, the key assumption of bagging is that the errors, that individual base learners make, are not all the same. If this holds true, the performance is always improved. In any case, the expected error of the ensemble cannot be higher than the expected error of the base learners. [8]

$$E_{baselearner} \leq E_{ensemble} \quad (2.2)$$

Slightly better performance can be achieved by employing random forests. When even higher improvements are desired, a method called *boosting* can be employed.

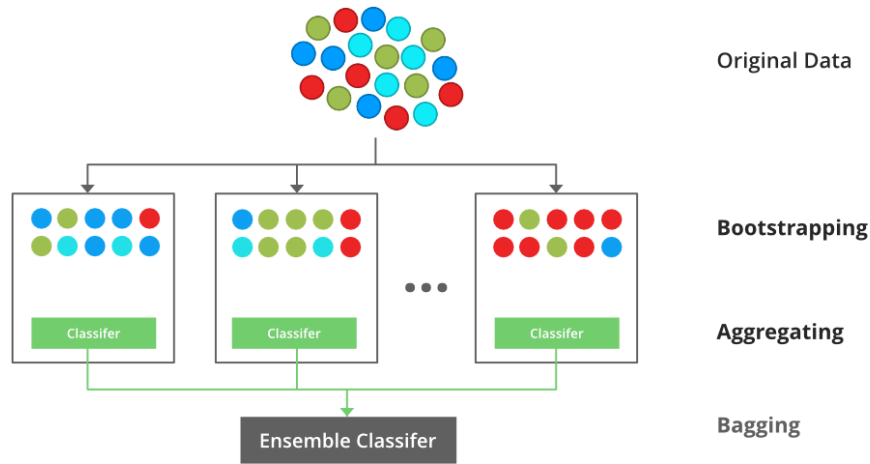


Figure 2.7: Bagging [6]

Variable Importance

Although bagging is usually more accurate than a single decision tree, one of the very appealing properties is lost, namely interpretation. Because when combining multiple base learners to create one strong learner, it cannot be simply illustrated as one single decision tree. In addition, finding out the most important variables in the procedure becomes a challenge. SSR can be used to obtain the feature importance for regression models and, for example, the Gini index can be applied to classification problems.

■ Random Forests

Random forests are a type of ensemble learning algorithm that combines decision trees using bagging. Random forests are an extension of bagging that makes the decision trees to be less correlated. [21]

Same as in bagging, subsets of data are created using bootstrapping and then used to train the base learners. The difference between bagging and random forests is that when training the decision trees, random forests pick a random sample of m predictors from all the features p as split candidates. A new sample of m features is determined at each new split, usually, $m = \sqrt{p}$, which provides a different subset of features for each of the base learners. This might not be very intuitive, but it solves the problem of one dominant feature that would be used in the majority of the base learners in the top split. As a result, the base learners (decision trees) using only the bagging method would be highly correlated; thus, their averaging would not yield the desired reduction in variance. However, since different subsets of features are forced, random forests overcome this issue. Typically, $(p - m)/p$ of the splits

do not even include the most influential feature which results in an overall decrease of the test error compared to bagging where $m = p$. [3]

2.3.2 Boosting

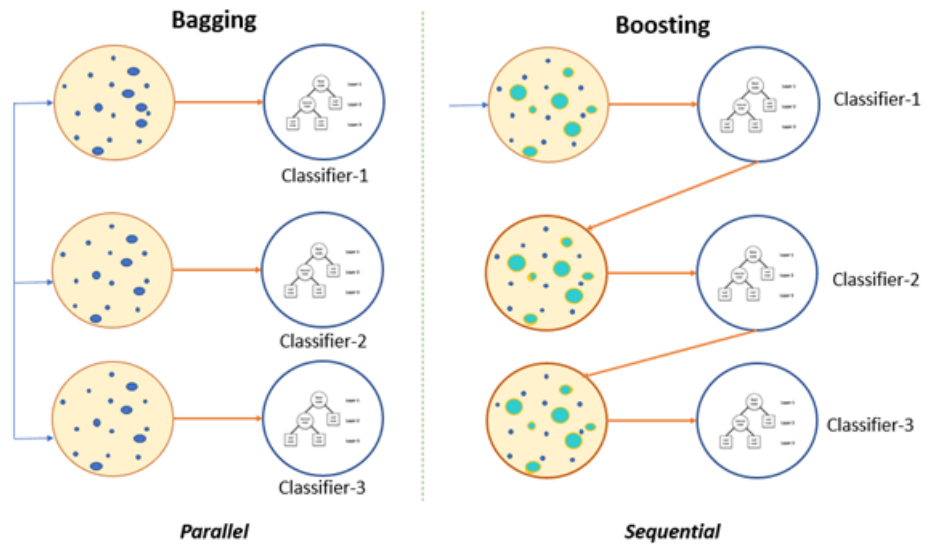


Figure 2.8: Difference between bagging and boosting [7]

The idea behind boosting is to train weak models in sequence, those models perform only slightly better than random guessing, and then iteratively improve the model by focusing on the examples that are misclassified by the previous model in the sequence. There are several different types of boosting algorithms, including AdaBoost, Gradient Boosting, and XGBoost. Each of these algorithms works by training a series of weak models and combining them to create a single strong learner. The base learners are typically decision trees, which are trained on different subsets of the original multiset. Boosting can produce highly accurate final models that are relatively insensitive to overfitting since they do not rely on one single model, but rather are a combination of several base models. [4]

The main difference between bagging, described in Section 2.3.1, and boosting is that in boosting, the weak learners are trained in sequence, and in bagging, the training of weak learners is performed in parallel (see Fig. 2.8).

AdaBoost

One of the oldest and the most used boosting algorithms is AdaBoost which stands for adaptive boosting. It was introduced in 1996 by Freund and Schapire. [8]

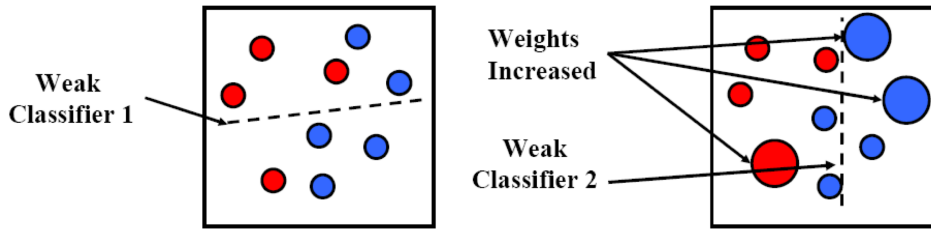


Figure 2.9: Boosting - increasing weights of misclassified data points [8]

Let's take a closer look at AdaBoost to demonstrate how boosting algorithms work. As already mentioned several times, boosting techniques train weak learners in sequence. In addition, each base learner is trained on a weighted dataset where the weights linked to each data point depend on the training of the previous classifier. When a data point is misclassified by the base learner, the weight associated with it is increased, as illustrated in Fig. 2.9. In the left part of the Fig. 2.9, the data points are divided into two groups (classes) by the first base learner, it is obvious that not all data points are classified correctly; hence, the misclassified points receive higher weights (marked by bigger circles on the right side of the figure) and with these weights, the second classifier proceeds with training and splits the data differently. This means that at each step, AdaBoost trains a new classifier that focuses more on the points that were previously incorrectly classified, through increased weights these points are prioritized in the next iteration. This process continues until all base learners are trained, then the final prediction is made through a weighted majority voting scheme. [8]

AdaBoost Algorithm procedure

Considering a two-class classification task, where x_1, \dots, x_N are input vectors, that can be assigned to one of two labels $t_n \in \{-1, 1\}$. Each data point receives a weight parameter w_n and weak classifiers $y_m(x)$ are trained on a weighted form of the inputs. [8]

1. Initialization of weight parameters $w_n = \frac{1}{N}$ for $n = 1, \dots, N$
2. For $m = 1, \dots, M$:
 - a. Minimizing the weighted error function by fitting a classifier to the training dataset

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(x_n) \neq t_n) \quad (2.3)$$

where $I(y_m(x_n) \neq t_n)$ is called an indicator function, it equals 1 when $y_m(x_n) \neq t_n$ and 0 otherwise.

b. Quantities calculation

$$\epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}} \quad (2.4)$$

and based on quantities ϵ_m , determine

$$\alpha_m = \ln\left\{\frac{1 - \epsilon_m}{\epsilon_m}\right\} \quad (2.5)$$

c. Updating weights

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(y_m(x_n) \neq t_n)\} \quad (2.6)$$

3. Final model as a weighted combination of base models y_m

$$Y_M(x) = \text{sign}\left(\sum_{n=1}^N \alpha_m y_m(x)\right) \quad (2.7)$$

Initially, weight parameters $w_n^{(1)}$ are equally distributed between train data points, and the first classifier $y_1^{(x)}$ is trained. After that, weight coefficients are recalculated based on equation 2.6 and either decreased for correctly classified data points or increased for incorrectly classified points. Error rates of each classifier are denoted by quantities ϵ_m which are then used to calculate weight parameters α_m for each of the base classifiers. The bigger the weight of the classifier, the more accurate results the base model yields. [8]

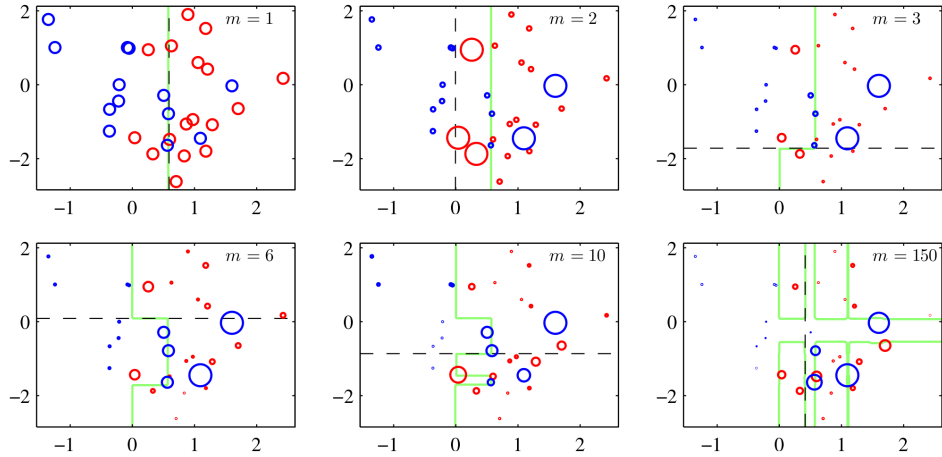


Figure 2.10: AdaBoost Algorithm [8]

Fig. 2.10 depicts the Adaboost algorithm, where base learners are decision stumps. Decision stumps are a type of decision tree with only one level, meaning the depth of the decision tree equals 1. In other words, a decision

stump is a tree with a single decision node and two leaf nodes. Predictions are based on whether a single feature exceeds a certain threshold or not and as consequence, the input space is partitioned into regions. Decision stumps tend to have high bias and low accuracy compared to more complex models. However, when used as part of an ensemble method, such as boosting, they can progressively contribute to the overall performance and accuracy of the final model. [8]

Chapter 3

Gradient Boosting Machine

Predicting energy consumption; and thus, estimating savings is crucial for evaluating the energy efficiency of buildings. Nowadays, the availability of countless data from smart devices in buildings with a combination of machine learning algorithms opens up new predicting possibilities. Very popular in building energy modeling are neural networks because of their capability of representing complex trends in energy consumption. Another famous approach is SVM¹ that can capture nonlinear trends using even quite small training datasets. Nevertheless, SVM requires choosing the right kernel and NNs are dependent on selecting the right topology; therefore, both of them are more difficult to tune than the gradient boosting machine (GBM) algorithm. GBM is a part of the boosting ensemble methods that keep delivering promising results. Weak learners (in this case decision trees) for GBM are generated sequentially and are dependent on each other, as explained in Boosting section 2.3.2 of Ensemble algorithms. Since the procedure is sequential, GBM is able to lower the bias, and because of the fact that each weak learner has a different contribution to the final strong learner, the variance is reduced. [12]

Boosting techniques were initially introduced for classification tasks. In 2001, Friedman developed the gradient boosting machine method as a boosting algorithm targeting regression problems. [9] The GBM algorithm begins by initializing (guessing) the first base learner (decision tree) and then at each following step, a new decision tree is built to minimize the loss function and appended to the model. The word "gradient" in gradient boosting refers to the gradient of the loss function that the algorithm is trying to minimize. The generated base learner at each step should be as correlated with the negative gradient of the loss function as possible so that the model is updated in the direction that reduces the loss function. GBM is very flexible since it depends on the choice of the loss function; thus, it can be customized to target a specific problem or application. There are a variety of loss functions that can be selected from or even application-specific loss functions can be developed. [12] [9]

The GBM algorithm is *stage-wise* which means that the previously added weak learners are not altered when the new decision tree is added. Moreover,

¹SVM = Support Vector Machines

each base learner is added with a certain weight parameter λ , also called the learning rate or shrinkage parameter, which forces the model to make smaller steps in the right direction to improve the model's performance and accuracy. The model can be also improved by adding randomization to the fitting process by subsampling. [12]

GBM has four hyper-parameters that have to be tuned [12]:

1. Depth of the decision tree ... d
2. Number of iteration/decision trees ... M
3. Learning rate/Shrinkage parameter ... λ
4. Subsample fraction ... η

3.1 Methodology - Function Estimation

Taking into consideration the function estimation task. Since GBM is a supervised ML technique, the dataset $(x, y)_{i=1}^N$ has to be provided with labels (also called target or dependent variables) y and input feature vectors x . The aim is to find the unknown function f that maps the corresponding labels y to the right feature vectors x . The estimated function is denoted as $\hat{f}(x)$ and it reduces some chosen loss function $\Psi(y, f)$ [9]:

$$\hat{f}(x) = y \quad (3.1)$$

$$\hat{f}(x) = \arg \min_{f(x)} \Psi(y, f(x)) \quad (3.2)$$

In equations 3.1 and 3.2, no assumptions about the functions f and \hat{f} are made; hence, it is non-parametric. The equation 3.2 can be rewritten using expectations [9]:

$$\hat{f}(x) = \arg \min_{f(x)} E_x[E_y(\Psi[y, f(x)]|x)], \quad (3.3)$$

where E_x is the expectation over the whole dataset and E_y is response variable (expected y loss). The target variable y can be derived from various distributions; therefore, different loss functions Ψ can be chosen. The function estimation becomes more manageable when some parameter is added to restrict the search space. Then the function estimation turns from non-parametric into parametric optimization: [9]

$$\hat{f}(x) = f(x, \hat{\theta}), \quad (3.4)$$

$$\hat{\theta} = \arg \min_{\theta} E_x[E_y(\Psi[y, f(x, \theta)]|x)], \quad (3.5)$$

where θ is the added parameter. Usually, in order to solve this parametric estimation, an iterative approach has to be taken. [9]

3.1.1 Numerical Optimization

The estimation of parameter θ can be formed in increments [9]:

$$\hat{\theta} = \sum_{i=1}^K \hat{\theta}_i, \quad (3.6)$$

where K is the number of iteration steps.

The steepest gradient descent method is often applied for parameter estimation. In the steepest gradient descent, the model's parameters are updated using the negative gradient of the loss function with respect to the parameters. The direction of the gradient signifies the greatest increase in the loss function, so taking the negative of the gradient points in the opposite direction, which is the direction of the greatest decrease in the loss function. [8]

The dataset $(x,y)_{i=1}^N$ is provided and the target is to minimize the empirical loss function $J(\theta)$ over the observations[9]:

$$J(\theta) = \sum_{i=1}^N \Psi(y_i, f(x_i, \hat{\theta})) \quad (3.7)$$

The steepest gradient descent optimization works by iteratively updating the parameter as it moves along the negative gradient of the loss function $\nabla J(\theta)$.

Notation [9]:

- $\hat{\theta}$... the parameter estimate
- $\hat{\theta}_t$... t-th incremental step of the parameter estimate $\hat{\theta}$
- $\hat{\theta}^t$... estimate of the whole ensemble = sum of all increments from the beginning of the optimization until the end

The steepest gradient descent optimization works as follows [9]:

1. Initialization of $\hat{\theta}_0$

For each step the following procedure is repeated:

- a. Getting collapsed parameter estimate $\hat{\theta}^t$

$$\hat{\theta}^t = \sum_{i=0}^{t-1} \hat{\theta}_i \quad (3.8)$$

- b. Assessment of the gradient of the loss function $\nabla J(\theta)$:

$$\nabla J(\theta) = \left[\frac{\partial J(\theta)}{\partial J(\theta_i)} \right]_{\theta=\hat{\theta}^t} \quad (3.9)$$

- c. Determination of the new estimate $\hat{\theta}_t$:

$$\hat{\theta}_t \leftarrow -\nabla J(\theta) \quad (3.10)$$

- d. The new estimate $\hat{\theta}_t$ is being added to the ensemble

3.1.2 Optimization in Function Space

In boosting techniques, the optimization is performed in the function space so that function estimate \hat{f} has an additive form [9]:

$$\hat{f}(x) = \hat{f}^M(x) = \sum_{i=0}^M \hat{f}_i(x), \quad (3.11)$$

where \hat{f}_0 is the initial estimation guess, K is the number of iteration steps, and $\{\hat{f}_i\}_{i=1}^M$ are boosts (function increments).

Following the same approach, base learners can be parameterized as $h(x, \theta)$ and then incrementation of base learners' functions can be employed using an optimal step-size ρ [9]:

$$\hat{f}_t \leftarrow \hat{f}_{t-1} + \rho h(x, \theta_t) \quad (3.12)$$

$$(\rho_t, \theta_t) = \arg \min_{\rho, \theta} \sum_{i=1}^N \Psi(y_i, \hat{f}_{t-1}) + \rho h(x_i, \theta) \quad (3.13)$$

3.2 Gradient Boosting Algorithm

Since the functions of the loss $\Psi(y, f)$ and the base learners $h(x, \theta)$ can be chosen or defined arbitrarily, the solution is quite hard to achieve; hence, a new function $h(x, \theta_t)$ can be selected in a way that it is the most parallel to the negative gradient along the observations $\{g_t(x_i)\}_{i=1}^N$ [9]:

$$g_t(x) = E_y \left[\frac{\partial \Psi(y, f(x))}{\partial f(x)} \middle| x \right]_{f(x) = \hat{f}^{t-1}(x)} \quad (3.14)$$

The simplification is that instead of searching for a general solution, the solution is obtained in a way where the increments are moving in the direction of $-g_t(x)$. This simplification allows to turn the optimization task into least-squares minimization [9]:

$$(\rho_t, \theta_t) = \arg \min_{\rho, \theta} \sum_{i=1}^N [-g_t(x_i) + \rho h(x_i, \theta)]^2 \quad (3.15)$$

This implies that the specific formulas are dependent on the choice of the loss function $\Psi(y, f)$ and the base learners $h(x, \theta)$.

GBM pseudo-code [9] [12]:

Inputs:

- Dataset $(x, y)_{i=1}^N$
- Number of iterations M
- Selected loss function $\Psi(y, f)$
- Selected weak learner model (decision trees) $h(x, \theta)$

Algorithm:

1. Initialization of \hat{f}_0
2. For each step $t=1, \dots, M$:
 - a. Calculation of negative gradient $g_t(x)$
 - b. Fitting a new weak learner $h(x, \theta_t)$
 - c. Obtaining optimal step size ρ_t :

$$\rho_t = \arg \min_{\rho} \sum_{i=1}^N \Psi \left[y_i, \hat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t) \right] \quad (3.16)$$

- d. Adjusting the estimation of the function:

$$\hat{f}_t \leftarrow \hat{f}_{t-1} + \rho_t h(x, \theta_t) \quad (3.17)$$

3.3 GBM Design

The GBM design is derived from the selection of the loss function $\Psi(y, f)$ and base learners $h(x, \theta)$ that are chosen to fulfill desired application.

3.3.1 Loss Functions

Generally, the loss function can be either chosen from an already developed loss function suitable for GBM or arbitrarily specified. The loss function can be split into three different groups based on the target variable y [9]:

1. *Categorical target variable, $y \in \{0, 1\}$:*
 - *Binomial loss function*
 - *Adaboost L_1 loss function*

2. Continuous target variable, $y \in \mathbb{R}$:

- Laplace L_1 loss function
- Gaussian L_2 loss function
- Huber loss function
- Quantile loss function

3. Other:

- Loss functions counts data
- Loss functions for survival models
- Arbitrarily defined loss functions
- ...

Since this thesis is focused on predicting the input power of buildings which is a regression task, only continuous loss functions are discussed.

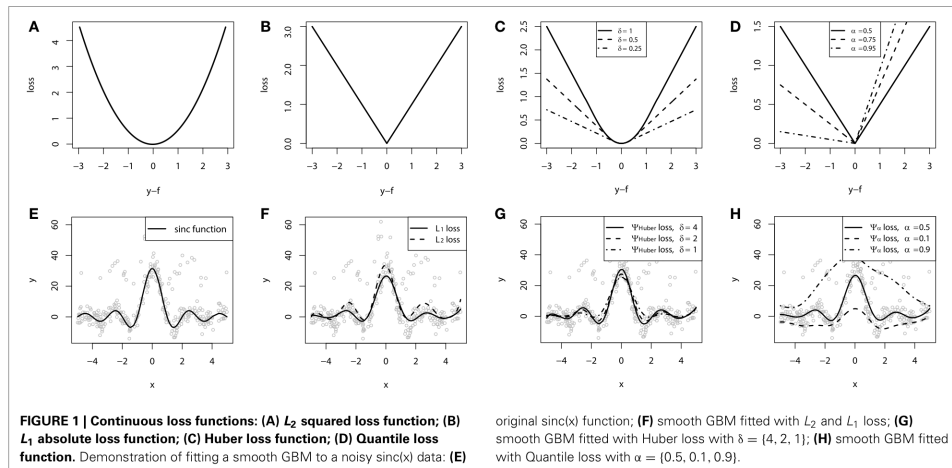


Figure 3.1: Loss functions for continuous target variable y [9]

Fig. 3.1 depicts L_1 , L_2 , Huber, and Quantile loss functions. The subfigures E-H are the applications of the loss functions to an artificially created dataset.

Gaussian L_2 loss function

L_2 loss function, also called the squared error loss function, is one of the most used ones. It is defined as [9]:

$$\Psi(y, f)_{L_2} = \frac{1}{2}(y - f)^2 \quad (3.18)$$

This loss function penalizes incorrect predictions made by the model. The higher the deviation from the dependent/target variable, the bigger the penalization; therefore, it is not very robust to outliers.[9]

Laplace L_1 loss function

Laplace L_1 loss function is more robust to outliers since they are not penalized as strongly as with the L_2 loss function. L_1 loss function is formulated as [9]:

$$\Psi(y, f)_{L_1} = |y - f| \quad (3.19)$$

Huber loss function

Another robust loss function is the Huber loss function, which is a combination of L_1 and L_2 loss functions. Its form is [9]:

$$\Psi(y, f)_{Huber, \delta} = \begin{cases} \frac{1}{2}(y - f)^2 & |y - f| \leq \delta \\ \delta(|y - f| - \frac{\delta}{2}) & |y - f| > \delta \end{cases} \quad (3.20)$$

where δ is the cutting-edge parameter that is able to control the robustness of the Huber loss function.

Quantile loss function

The quantile loss function is general, distribution-free, and robust to outliers. It is designed as follows [9]:

$$\Psi(y, f)_\alpha = \begin{cases} (1 - \alpha)|y - f| & y - f \leq 0 \\ \alpha|y - f| & y - f > 0 \end{cases} \quad (3.21)$$

where α is a parameter that controls the conditional distribution. When $\alpha = 0.5$, the Quantile loss function would equal the Laplace loss function.

■ 3.3.2 Weak Learners

GBM can be implemented using several types of weak learners. The commonly used base learners are [9]:

- *Linear models*
 - *Ordinary linear regression*
 - *Ridge regression - penalized linear regression*
 - *Random effects*
- *Smooth models*
 - *Radial basis functions*
 - *P-splines*
- *Decision Trees*
 - *Decision stumps*
 - *Decision trees with different depths*

- *Some other models*

- *Wavelets*
- *Markov random fields*
- *Base-learner functions (tailor-made)*

GBM model can either use only one type of base learners (homogeneous ensembles) or even combine multiple ones (heterogeneous ensembles).

In this work, only **decision trees as weak learners** are explored. They can capture non-linear relationships and are suitable for both categorical and continuous variables. Decision trees partition the feature space into rectangles, where each split is associated with one if-then-else rule. For a more detailed explanation (see Chapter 2.2 about decision trees).

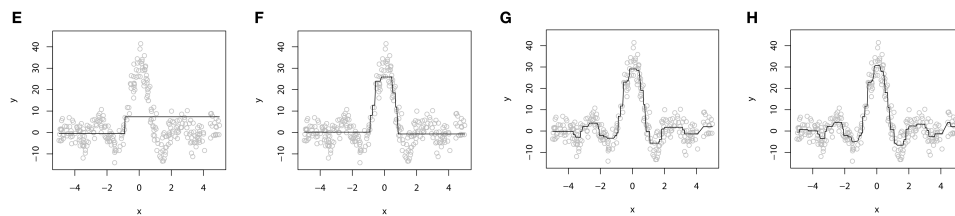


Figure 3.2: Decision trees as weak/base learners [9]

As illustrated in Fig. 3.2, the choice of the number of iterations M matters a lot as it controls the accuracy of the final model. When it comes to the loss functions, it is advised to start with the L_2 loss function and then move to the more complex ones. For the base learners, decision trees with small depth or decision stumps are commonly used. [9]

Since decision trees are prone to overfitting, the GBM model with decision trees as base learners is also susceptible to overfit the training data. There are several regularization approaches that can be implemented to prevent the undesired overfitting tendency.

■ 3.3.3 Regularization Techniques

One of the main aspects of a successful ML model is that it can generalize well and have a good performance on both the training and test data (the newly presented data). Regularization methods can be deployed to restrict the fitting process so that overfitting is reduced. [9]

From Fig. 3.3, it is noticeable that the model overfits to the data, it has learned the training data perfectly including the noise, which is not desirable because the model then cannot make reasonable predictions for data that it has not seen before. The Fig. 3.4 represents different combinations of the

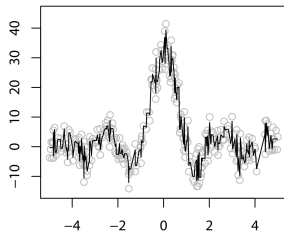


Figure 3.3: GBM regression overfitting example [9]

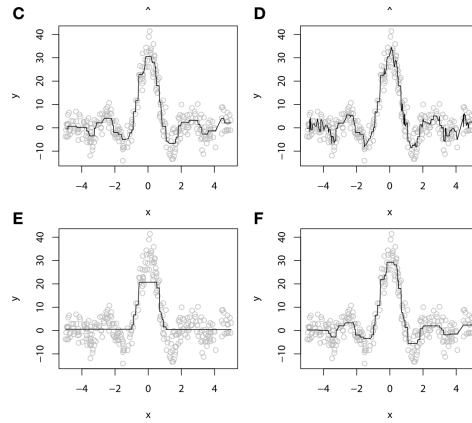


Figure 3.4: Fitting a decision-tree GBM to a noisy data [9]

number of steps/iterations M and the learning rate λ : **C)** $M = 100, \lambda = 1$; **D)** $M = 1000, \lambda = 1$; **E)** $M = 100, \lambda = 0.1$; **F)** $M = 1000, \lambda = 0.1$.

Popular GBM regularization techniques [9]:

- *Subsampling*
- *Shrinkage*
- *Early Stopping*

Let's briefly explain each of them.

■ Subsampling

Generally, subsampling has shown enhanced generalization ability and reduced time and computational demands. Instead of training the weak learner using the whole dataset, a subset of data is sampled without replacement (typically) and used for training. Subsampling needs the subsample fraction η , also called bag fraction, to be picked. For instance, if the subsample fraction is set to 0.5, it means that only 50% of the initial data points are used for training each decision tree which not only elevates the accuracy but also reduces the computational demands since less data overall is used. The $\eta = 0.5$ gives meaningful outcomes when the data is abundant. When insufficient training data is available, the subsample fraction must be chosen carefully. [12] [9]

■ Shrinkage

Shrinkage is implemented in order to decrease the impact of each gradually added decision tree. It reduces the effect of each iteration because improving models by taking many little steps yields much better results than taking

fewer bigger steps. The shrinkage parameter λ , also known as learning rate, can take on values between 0 and 1, $\lambda \in (0,1)$. The lower the value of λ , the better generalization is reached; however, more steps must be taken for the algorithm to converge. [12] [9]

The shrinkage regularization method is executed in the final iteration of the GBM algorithm [9]:

$$\hat{f}_t \leftarrow \hat{f}_{t-1} + \lambda \rho_t h(x, \theta_t) \quad (3.22)$$

As demonstrated in Fig. 3.4, shrinking the learning rate λ implies that more iteration M have to be taken to reach the desired accuracy. Nevertheless, using the shrinkage parameter enables the model to learn more details, which is especially useful for decision trees because they cannot represent details very well. [9]

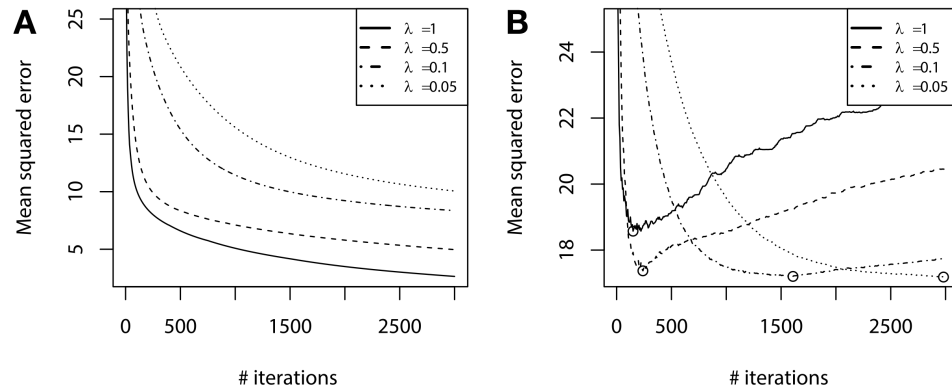


Figure 3.5: Training set error vs. Validation set error [9]

But how do the number of iterations M and the value of the learning rate λ impact the effect of overfitting? Fig. 3.5 compares how setting different λ values affects the number of iterations and the overall error evaluated on the training dataset (Fig. 3.5A) and on the validation set (Fig. 3.5B). As can be observed from the curves in Fig. 3.5A, the speed of convergence is highly affected by the learning rate. However, the more important factor is the behavior of the validation set error, depicted in Fig. 3.5B. The minimum error value for each chosen λ value is denoted as a circle, which shows that the lower the value of learning rate λ , the lower the error of the validation set, but the more steps have to be taken. [9]

In summary, increasing the number of iterations too much can lead to overfitting and a very small value of M may, on the other hand, stop the training process too early causing the final model to perform poorly. When considering the learning rate, larger values of λ can cause overfitting. On the contrary, a smaller learning rate can slow down the training process and may result in

underfitting if it takes too long to reach the optimal parameters. To achieve the optimal behavior of the final GBM model, both of these parameters have to be tuned carefully. Some methods can be implemented to find the right combination of parameter values.

■ Early Stopping

Considering the regularization techniques discussed above, when the learning rate λ is given, the optimal number of steps M_{opt} typically does not equal the initially set M . It would be ideal to stop adding decision trees to the ensemble when the minimum validation error is reached. Early stopping is implemented by training the model on a training set and periodically evaluating the model's performance on a separate validation set. The training process is stopped when the model's performance on the validation set starts to degrade, as this indicates that the model begins to overfit the training data. [9]

■ 3.4 GBM Interpretation

Interpreting the results of GBM can be challenging, as the final model is a complex ensemble of base learners (decision trees), it cannot be easily visualized as one single decision tree. There are several techniques that can be used to interpret and understand the results of a GBM model. Let's take a closer look at two popular interpretation techniques. [9]

■ Relative Variable Influence

Relative variable influence is used to examine the importance of features used by the model. Every split on a feature (variable) in a decision increases the likelihood of the whole ensemble, and the sum of log likelihood across all decision trees grows. [9]

The variable influence is defined as [9]:

$$Influence_j(T) = \sum_{i=1}^{L-1} I_i^2 1(S_i = j), \quad (3.23)$$

where L is the number of tree splits. The sum is until $L - 1$ because all the nodes except the leaves are of interest. The variable influence equation 3.23 is based on how many times each feature is chosen for splitting. S_i is the current splitting variable, j is the queried splitting feature, and I_i^2 denotes the weights of the influence. The influence of feature j in the whole ensemble is the average of the variable influence in each decision tree [9]:

$$Influence_j = \frac{1}{M} \sum_{i=1}^M Influence_j(T_i) \quad (3.24)$$

The influences are then scaled so that they sum up to 100%. [9]

■ Partial Dependence Plots

Visualization is a very useful way of interpretation. Partial dependence plots show the effect of a feature on the model's predictions after marginalizing out all other explanatory variables. This can be useful for understanding how a particular feature is influencing the model's predictions. Even though these plots might not depict the effects perfectly, they can be a good foundation for interpretation. Pairwise dependence plots can be convenient to visualize relationships between couples of variables. [9]

Chapter 4

Model Implementation

The input power and energy consumption of large buildings depend on multiple factors that are often complex and diverse. There are three main groups that the factors can be grouped to [22]:

- *Physical and technical factors* → building design and climate
- *Occupancy* → number of occupants and their activities
- *Social and economic factors* → diverse energy usage in buildings

The necessary data for this master thesis was provided by the company Energocentrum plus s.r.o. The data is collected in hourly or 15-minute intervals from smart commercial buildings in Prague. Each building's dataset includes information about the timestamp, occupancy of the building, outside temperature, and input power.

The ability to obtain high-interval data from smart meters helps to create better predictive models with accurate forecasts. In general, the more data is available, the better the model generalizes and its performance improves. However, as a consequence, processing large amounts of data is very time- and memory-consuming. Hence, finding some sort of compromise between the data amount and the computational cost is desired.

Let's take a closer look at the supplied input data and preprocessing methods that are deployed to modify the inputs.

4.1 Data Preprocessing

It is sometimes surprising how much time and effort data preparation consumes. To build a successful model, the data needs to be gathered, cleaned, preprocessed, and only after that it can be used for model training. Building an ML model is an iterative process, where the model is trained, and the results are evaluated. Based on the analysis of the outcomes, the input data can be modified, new features can be added or the model itself can be tuned. [23]

Table 4.1: Example - Data for building number 29

ts	occ	temp	meas
2019-01-01 00:00:00	True	0.0	NaN
2019-01-01 01:00:00	True	0.0	0.0
2019-01-01 02:00:00	True	0.0	1.3
2019-01-01 03:00:00	True	0.0	1.3
...
2022-10-25 21:00:00	True	17.0	5.25
2022-10-25 22:00:00	True	18.0	4.84
2022-10-25 23:00:00	True	19.0	3.66
2022-10-26 00:00:00	True	19.5	6.33

that is intended to be predicted by the model. Since the target variable is a continuous variable, the whole task is a regression problem.

Visualization of the dataset for building No. 29 is depicted in Fig. 4.1, the data is recorded from the beginning of the year 2019 until the end of October 2022. The topmost graph depicts the values of temperature in °C, the trend is very clear and it is the same for all the buildings because the temperatures rise as summer approaches and they are the lowest in winter months. The input power in kW is illustrated in the middle plot, the building No. 29 shows quite consistent input power throughout the years, there are no clear signs that the building would spend significantly more energy on heating or air conditioning. These differences are discussed further down. And the last bottom plot shows on which days the building was occupied (denoted by 1) and unoccupied (denoted by 0). The vertical dashed-lined pictures the split between the train and test datasets, detailed explanation of the process is in Section 4.1.8.

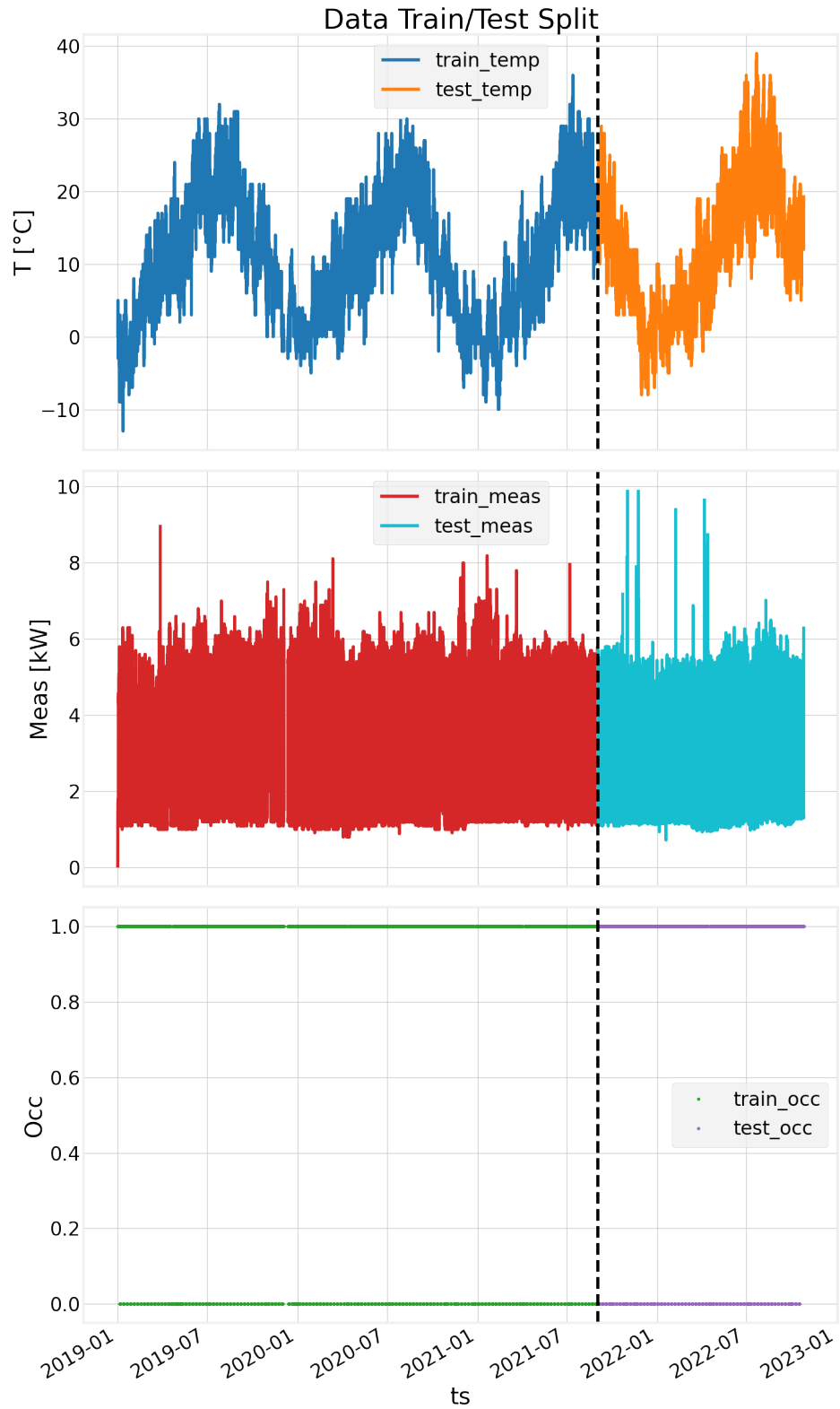


Figure 4.1: Dataset No. 29

■ Data Analysis

Inspecting the data visually is a crucial step in understanding the data, its structure, characteristics, and trends; therefore, implementing different sorts of plots to display the data is desired. By observing the graphs, some errors in the input data can be discovered and removed or modified; otherwise, they would yield unreasonably bad results.

Plots used for visualization of the data [25]:

■ Histogram

- shows a variable's distribution as a set of adjacent rectangles on a data chart
- represents counts of data within a numerical range of values

■ Scatter plot

- useful for exploring interrelations or dependencies between two different variables
- ideal for detecting outliers and trends in data

■ Box plot

- useful for seeing a variable's spread
- helpful for visually spotting outliers

■ Pie chart

- uses a circle divided into sections (categories) as percentages of a whole

■ Pair plot, also known as scatter plot matrix

- displays multiple pairwise scatter plots in a matrix format
- helpful for visualizing relationships between multiple variables in a dataset, potentially spotting any correlations
- useful to get a sense of the distribution of data

After visualizing pandas data frames for each building, different trends of input power can be spotted. As mentioned above, Fig. 4.1 shows quite consistent values of input power. However, some buildings use more energy in summer for air conditioning, as expressed in Fig. 4.2, and some others, on the contrary, use more energy for heating in winter, which is captured in Fig. 4.3.

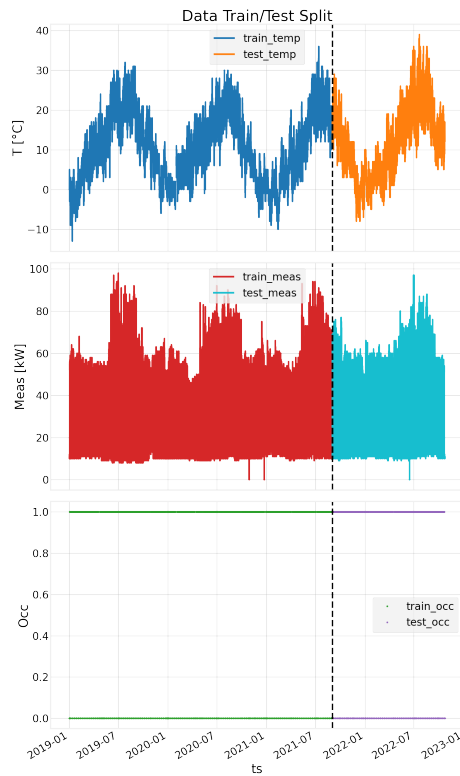


Figure 4.2: Dataset No. 33

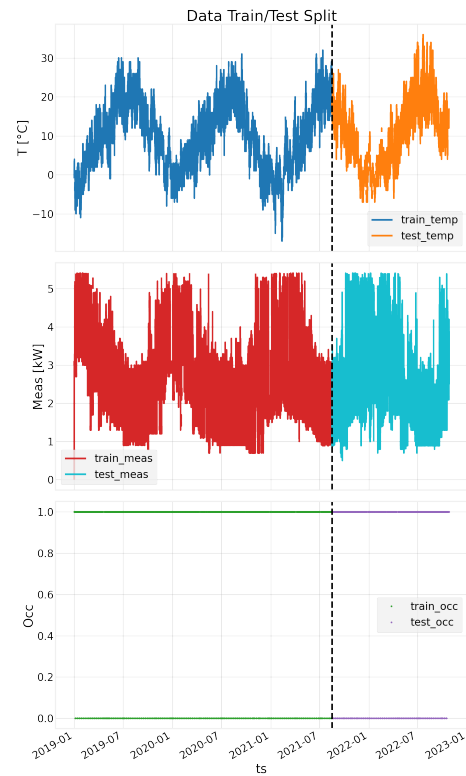


Figure 4.3: Dataset No. 63

By visual inspection and running scripts to detect unusual values, some anomalies are detected. Pie charts for the occupancy feature are generated to show the percentage of occupied versus unoccupied days for each building. When going through all the pie charts, it is clear which buildings work in continuous mode, meaning that the occupancy of such buildings is equal to 1 (True) all the time. An example of a pie chart for a building that is non-stop in operation depicts Fig. 4.4. There are only 6 buildings in the whole input data that exhibit continuous working mode.

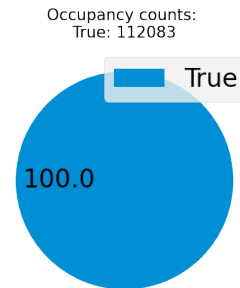


Figure 4.4: Occupancy pie chart for building No. 7

In the next step, the attention is focused on the values of temperature. The buildings that have values of temperature either below -25°C or above 50°C are filtered out since those temperatures would be considered extreme and unusual in Prague. Such temperature boundaries are exceeded only for 2

buildings, with indices 14 and 15, for which the measured temperatures reach more than 1000°C which is, of course, impossible, see Fig. 4.5. Another common issue is the presence of outliers (extreme values) that are usually caused by a momentary failure of the sensor or some power outage. Fig. 4.6 illustrates the existence of outliers in the measured input power for building No. 22, some of the outliers are so huge that the distribution of the data cannot be observed. Outliers can very negatively affect the prediction results, especially when the chosen loss function is not robust to outliers, e.g. squared loss function which is commonly used for regression. Thus, a good practice is to remove outliers before feeding the data into the model.

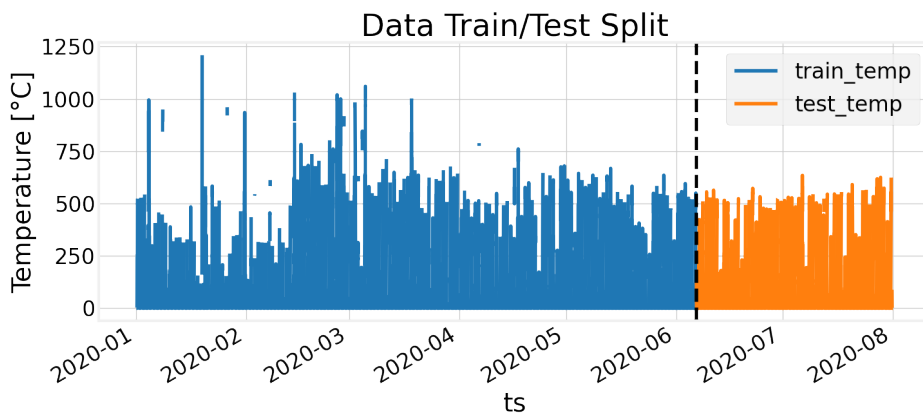


Figure 4.5: Incorrect values of temperature for building No. 14

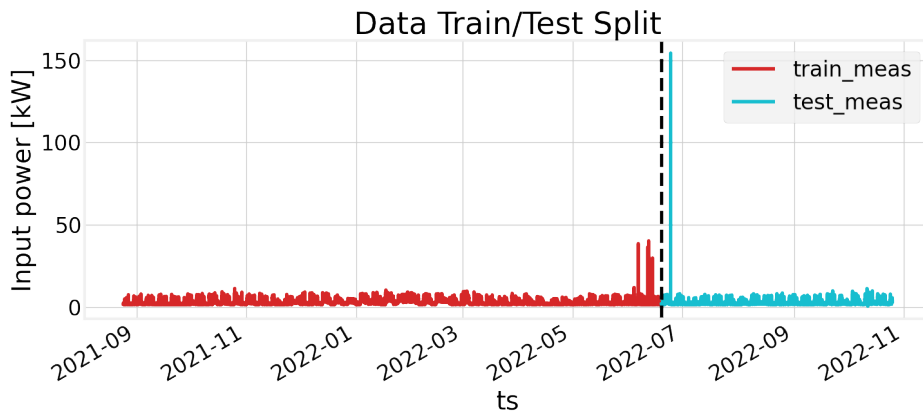


Figure 4.6: Example of the dataset with outliers (building No. 22)

■ 4.1.2 Handling Missing Values

Most of the ML models are sensitive to missing values, so it is important to treat them properly. What to do with missing (NaN¹) values is application dependent. Either the rows with missing values can be dropped from the dataset, or the missing values can be filled in with an appropriate replacement, for example, mean, median, or any specified value, etc. In the case of this work, all rows with missing values are dropped as the model cannot properly learn from them. Usually, the most missing values are in the column of a target variable of input power and there is no reason for keeping the rows with missing target variables because Gradient Boosting is a supervised learning method, so the labels (target variables) are necessary.

■ 4.1.3 Adding Features

To make the model more precise, additional features are extracted from the timestamp and appended to the data frame of each building, namely:

- *Hour*
- *Minute*
- *Day of week*
- *Quarter*
- *Month*
- *Year*
- *Day of year*
- *Day of month*
- *Week of year*

The names of the "datetime" features are self-explanatory and have integer values. These additional features deduced from the input data give the model extra information from which it can learn a certain pattern. Not always all added features have to be relevant, some can be highly correlated with others which would not provide any new information. Moreover, appending a redundant number of correlated features can even cause the curse of dimensionality. Hence, some feature selection techniques should be always implemented to find the most relevant features.

¹NaN = Not a Number

4.1.4 Transforming Categorical Features - Encoding

Even though ML models that are based on decision trees should be able to handle categorical features, it is always a good idea to encode categorical variables to prevent any issues. The only categorical feature in the input data is occupancy which can acquire values of either *True* or *False*. Encoding categorical features, like color, gender, or occupancy of a building, means transforming the string values into numerical values. **Label-Encoding** and **One-Hot-Encoding** are two popular approaches used for it.

The process of label encoding involves assigning a unique integer value to each category in a categorical variable. For example, consider a dataset with a categorical variable "color" that can take on the values "purple", "pink", and "blue". Using *LabelEncoder* from the scikit-learn library in Python, 0 would be assigned to "blue", 1 to "pink", and 2 to "purple". This label encoder works in a way that it sets integer values starting with 0 to the categories based on alphabetical order. This allows the machine learning algorithm to understand the ordinal relationship between the categories. It is important to note that label encoding does not create a numerical relationship between the categories; it only assigns an arbitrary integer value to each category. Algorithms based on decision trees are not affected by this arbitrary assignment. However, some other ML algorithms like linear regression and neural networks can assume a hierarchical relationship. To overcome this issue, one-hot encoding should be applied. [26]

One-hot encoding is another method used to convert categorical variables into a numerical form that can be provided as input to machine learning algorithms. One-hot encoding prevents any assumptions about the order in the encoded values because for each category it creates a new binary column. Considering the same "color" example as before ("purple", "pink", and "blue"), three new columns would be created, one for each color category, see Table 4.2. One-hot encoding will create a lot of new columns and might lead to high dimensionality if there are many categories. [26]

Table 4.2: One-hot encoding

Color	Color-purple	Color-pink	Color-blue
Purple	1	0	0
Pink	0	1	0
Blue	0	0	1

Since the occupancy feature is binary by itself, there is no reason to use one-hot encoding which would create one redundant column and increase the dimensionality for no benefit. Therefore, *LabelEncoder* from the scikit-learn library is used for purpose of encoding this categorical feature. The single problem was when the building has only *True* values for occupancy (non-stop in operation). Then, these *True* values would be encoded as 0 since the encoder always starts assigning values from 0; therefore, these cases are

identified and relabeled so that the values are consistent: *False* is mapped to 0 and *True* is encoded to 1.

4.1.5 Removing Outliers

Handling outliers is a key component of preprocessing because they can skew the distribution of the data and affect the statistical values such as mean, standard deviation, variance, etc. When the outliers are not removed and fed to the model, the model's results can lead to misleading conclusions. Nevertheless, removing outliers is not always a good idea, especially when the outlier values are meaningful for the model/analysis; therefore, visualization of the data can help to understand the appearance of outliers and treat them accordingly. There are multiple approaches how to handle outliers. In the case of gradient boosting methods, the effect of outliers can be also controlled by the choice of the loss function, for example, the Huber loss function is more robust to outliers than the most commonly used squared error loss function, for more information on loss functions (see Section 3.3.1). However, to obtain adequate results, it is essential to remove the outliers from the target variable (input power).

For outlier removal, **Interquartile Range (IQR)** method is deployed. This method uses the concept of quartiles, which divide a dataset into four equal parts. Let's explain this technique on the input power feature of building No. 22 that is displayed in Fig. 4.6. Using the pandas function *describe()* on the dataset, descriptive statistics are obtained (see Table 4.3). The 25% signifies the first quartile Q1 and the 75% value indicates the third quartile Q3. [27]

Table 4.3: Descriptive statistics before and after outliers removal using IQR

	Before outlier removal	After outlier removal
count	40968	39335
mean	2.894922	2.722410
std	1.561004	1.060807
min	0.590000	0.590000
25%	1.920000	1.920000
50%	2.280000	2.200000
75%	3.530000	3.360000
max	154.350000	5.920000

The IQR method works as follows using the data from Table 4.3 [25]:

1. Calculate the interquartile range by subtracting Q1 from Q3.
IQR = the distance between the first quartile (at 25%) and the third quartile (at 75%)
 $IQR = 75\%(value) - 25\%(value) = 3.53 - 1.92 = 1.61$

2. The data points that are more than 1.5 times the IQR below Q1 or above Q3 are considered outliers.

$$1.5 \cdot IQR = 1.5 \cdot 1.61 = 2.415$$

$$Q1 - 1.5 \cdot IQR = 1.92 - 2.415 = -0.495$$

$$Q3 + 1.5 \cdot IQR = 3.53 + 2.415 = 5.945$$
3. Remove the identified outliers from the dataset.

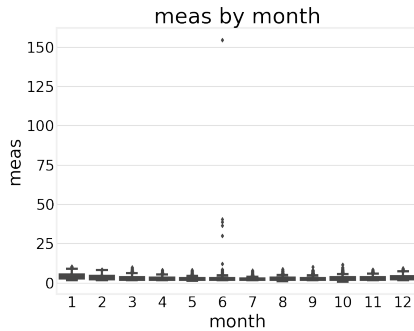


Figure 4.7: Box plot for building No. 22 before removing outliers

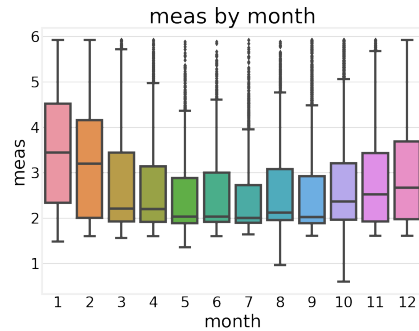


Figure 4.8: Box plot for building No. 22 after removing outliers

The visualization of box plots for building No. 22 can be observed in Fig. 4.7 (still with outliers) and in Fig. 4.8 (without outliers). These boxplots show the range of input power for each month. As can be deduced from the left Fig. 4.7, the biggest outliers occur in June, maybe there was some problem with a particular smart sensor and thus, such extreme values were measured. When the outliers are removed, the final distribution of the input power distribution is plotted in Fig. 4.9. The y-axis called meas corresponds to the input power in kW.

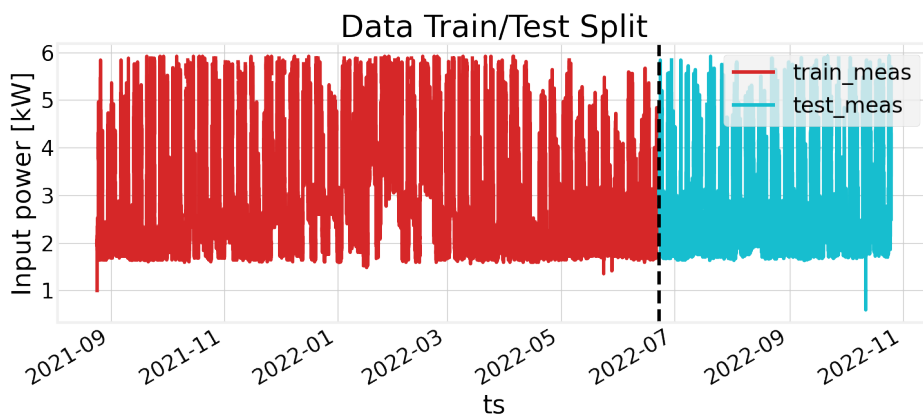


Figure 4.9: Dataset No. 22 after outlier removal

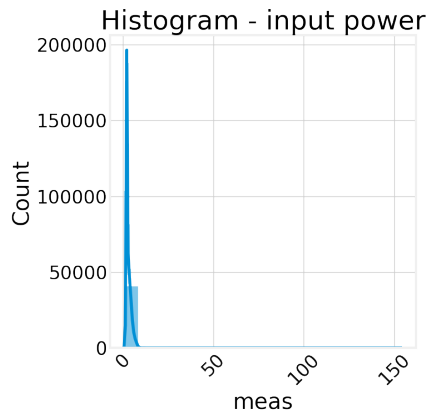


Figure 4.10: Histogram for building No. 22 before removing outliers

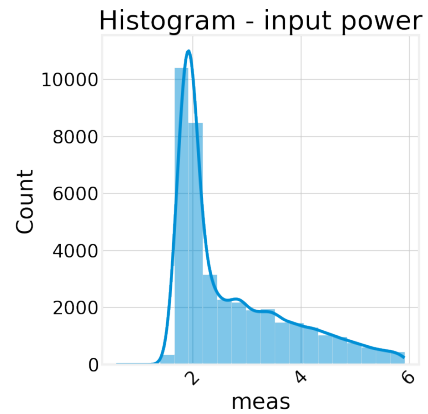


Figure 4.11: Histogram for building No. 22 after removing outliers

The differences in the values and the kernel density distribution of input power before and after the outlier removal can be also observed in Fig. 4.10 and Fig. 4.11. These figures are histograms with KDE² denoted by the solid line.

4.1.6 Scaling

Scaling is an important step in the preprocessing of data because some ML algorithms are sensitive to the magnitude of features which can cause the feature to be more dominant than others. In general, scaling is not necessary for gradient boosting algorithms, such as XGBoost, which are based on decision trees, so they can handle data of different ranges. However, it might still improve the performance of the learner and lead to faster convergence. In addition, when the data is scaled, it is much easier to compare error metrics, like mean squared error, among the buildings. The performance of the model is tested when the data is scaled and when it is not and the conclusion is that scaling slightly improved the accuracy; thus, it stays employed in the data preparation process.

MinMaxScaler from the scikit-learn library is used to perform the feature scaling. *MinMaxScaler* transforms the minimum value of the feature to 0 and the maximum value to 1, the values in between are scaled accordingly. The shape of the original distribution is preserved. [28]

This work implements only scaling of temperature ('temp') and input power ('meas') features because the rest of the features (minutes, hours, days, months, etc.) have always the same range for all the buildings, so scaling those features does not bring any enhancements.

²KDE = Kernel Density Distribution

4.1.7 Treating Correlated Features

There are many techniques to visualize correlated features in a dataset, **heatmaps** and **pair plots** are one of those frequently used in Python, they can be easily implemented using the seaborn library.

Heatmaps can be used to illustrate the relationship between two or more variables, they provide the correlation matrix showing the direction of the correlation and how strongly are two features correlated. Heatmaps are based on Pearson correlation coefficients which are mainly intended to discover linear relationships between numeric variables. The correlation coefficients range anywhere between values -1 and 1. [29] [22]

Pearson Correlation Coefficient [25]:

- $R = 1$ → Strong positive relationship
- $R = 0$ → Not linearly correlated
- $R = -1$ → Strong negative relationship

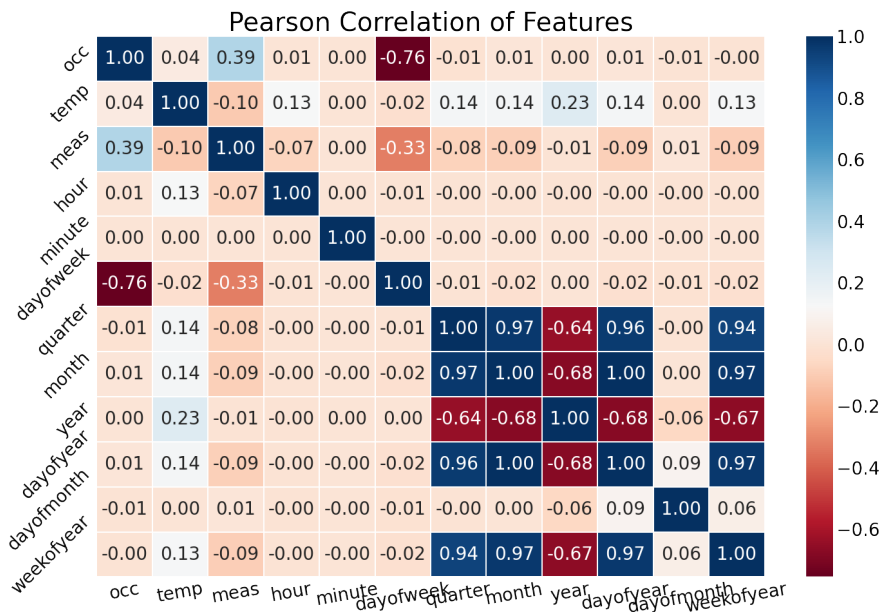


Figure 4.12: Heatmap for building No. 22 before dropping correlated fetures

Again, considering the dataset for building No. 22., Fig. 4.12 depicts the dataset with all the features where the numbers and colors denote how strongly and whether negatively or positively the features are correlated. The features that have values of the Pearson correlation coefficient either close to 1 or close to -1 show a high correlation with another feature and hence, should be removed. Fig. 4.13 captures the heatmap after dropping the

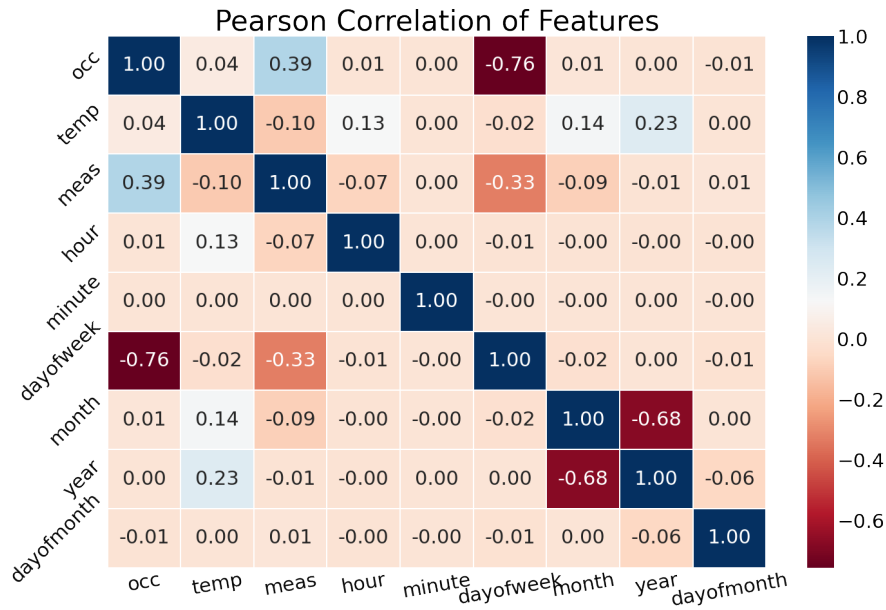


Figure 4.13: Heatmap for building No. 22 after dropping correlated fetures

most correlated features, those features are 'quarter', 'weekofyear', 'dayofyear' because they are very correlated with 'month'. The highly correlated features do not bring additional information to the model; moreover, they increase the dimensionality of the dataset.

Pair plots enable the visualization of relationships between multiple variables in a dataset in one figure. They plot all possible pairs of features against each other in a scatter matrix, the non-diagonal plots represent the relationship between two variables. The diagonal plots show histograms or kernel density estimates, which represent the distribution of each individual variable. Pair plots are not only useful in identifying correlations and associations between variables, but they can also help to uncover patterns and outliers. [30] [25]

Both heatmaps and pair plots make sense only for fairly small amounts of features; otherwise, the visualization would not be readable. Thus, the pair plot for building No. 22 in Fig. 4.14 shows only chosen features so that it is easily visible. The red color represents the data points that belong to the occupied days of the building and the blue color corresponds to the data for unoccupied days.

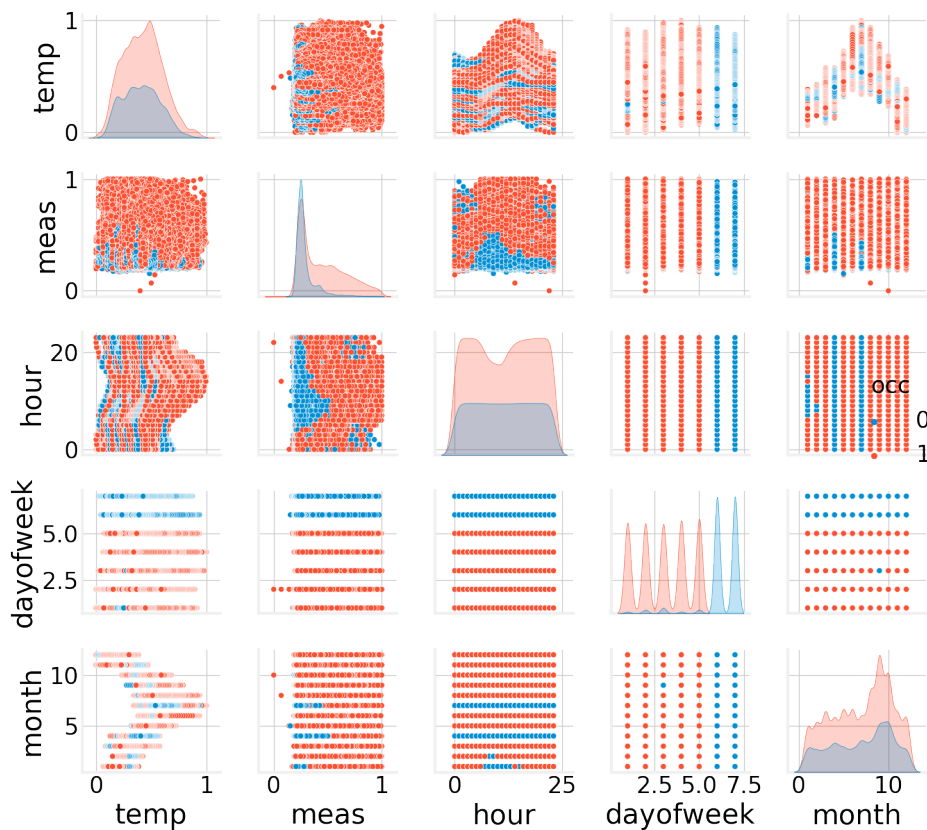


Figure 4.14: Pair plot for building No. 22

4.1.8 Splitting The Data

Diving the original data into a training and a test set is a key step in the ML process, especially for performance evaluation. The goal of any ML model is to be able to generalize well and give accurate results for new data points. However, when the model's performance is assessed using the same data that is used for training, the results might be overly optimistic because the learner is evaluated using the same data points as for training. In addition, such a model is prone to suffer from overfitting.

By splitting the data into training and testing datasets, the learner is trained on the training set, and its performance is then evaluated on the test (unseen) data to avoid any misleading conclusions. Thus, the common practice is to divide the available data into two subsets (train and test), it can be easily implemented using `train_test_split()` function from the scikit-learn library. For supervised learning methods, the data is further split into independent variables (features) and dependent variables (labels):

- *Train set* - usually contains around 70-80% of the initial data
 - *X_train* - features used for training

Important parameters to take into consideration when developing the gradient boosting regressor [32]:

- *Loss function* → specifies the loss function that is being optimized by the model
Possible loss functions: squared error, absolute error, Huber, quantile
- *Learning rate* → controls the contribution of each decision tree
Smaller steps mean slower convergence, but the algorithm is less likely to overshoot the optimal solution.
- *Number of estimators* → defines the number of trees in the ensemble
- *Subsample fraction* → identifies the fraction of samples used for fitting one individual weak learner
Subsample fraction must be chosen from (0,1) range.
- *Maximal depth* → controls the maximum depth of the decision trees
Large depths can capture more complex relationships, but also increases the risk of overfitting.

There are more parameters that can be tuned, but these ones make significant differences in performance. The overall performance of this gradient boosting regressor is later tested against the XGBoost implementation, for comparison results see Chapter 5.

■ 4.2.2 XGBoost Model

XGBoost stands for Extreme Gradient Boosting and was proposed by Chen and Guestrin at the University of Washington in 2014 [13]. XGBoost is similar to GBM; however, it is an optimized version of GBM that delivers even more promising results and operates faster. By adopting advanced regularization, XGBoost generalizes better and hence, prevents overfitting. It uses the second-order derivative of the loss function and due to the possibility of parallel computing, the training of the model is faster than other popular approaches, the newer libraries such as LightGBM and CatBoost might yield even faster results. Compared to the gradient boosting scikit-learn library, XGBoost can even handle missing values, is more efficient, provides a more extensive set of hyperparameters, and also have support for regularization. The XGBoost library has available the *XGBRegressor* class that can be used for solving gradient boosting regression problems. [21] [33]

There are more available parameters than for the scikit-learn implementation; however, the main ones remain the same. XGBoost regressor is a slightly more advanced model than the previous gradient boosting which results in more variability in tuning the hyperparameters. Some principal parameters that affect the model's accuracy and training speed are [31]:

by automatically stopping the training process when the performance of the model on a validation set stops improving for a specific number of rounds. If the number of early stopping rounds parameter is set to a high value, the model is trained for more rounds even when the performance decreases and has more chances to overfit the data. On the contrary, if the value is too low, it might stop the training process too early and prevent possible further improvements. [31]

When running a comparison analysis of the previously mentioned *Gradient-BoostingRegressor* model in Section 4.2.1 against the *XGBRegressor* model. The performance of both models is mostly similar; however, when it differs, the *XGBRegressor* usually shows at least slightly better results, more about that in Chapter 5. Because the *XGBRegressor* model yields better results and the XGBoost library is more powerful, further parameter tuning and the model's accuracy enhancements are performed only on the *XGBRegressor* model.

■ XGBoost Model Performance Improvements

There are several things that might improve the model's performance. The universal preprocessing steps are already discussed in Section 4.1. All the data preparation steps such as adding "datetime" features, removing outliers, scaling, and dropping correlated features boost the model's predicting capabilities. For instance, the accuracy of predictions for the dataset of building No. 22 that initially had big outliers increased from 26% to 70% just by removing the outliers. However, the preprocessing actions do not improve the predictions for all of the buildings, but overall performance rises.

The additional potential enhancements that are explored and tested in this work are:

1. *Incorporating public holidays*
2. *Adding average features*
3. *Adding lag features*
4. *Relabeling*
5. *Changing the split size between train and test sets*
6. *Altering the subsample fraction*
7. *Choosing a suitable loss function*

Each of the possible enhancements is implemented and tested on a subset of buildings to see how the change affects the R-squared score (R^2) and the MSE⁴; the ones that improved the accuracy stay implemented, the rest are excluded.

⁴MSE = Mean Squared Error

Incorporating public holidays

The idea here is to check whether the building is mostly in constant operation by calculating the percentage of unoccupied days from the whole data set. If the percentage of unoccupied days is less than 20%, keep the occupancy values as they are. On the other hand, when the percentage of unoccupied days is more than 20%, meaning that the building does not operate nonstop, then it is assumed that during public holidays the building is empty (unoccupied). Therefore, the days that fall on Czech public holidays are set to unoccupied. This change is tested on 4 random buildings that have no unoccupied days outside of weekends and the predicting accuracy (R^2 score) is increased as can be observed in Table 4.4.

Table 4.4: Performance improvement when public holidays incorporated

Building index	R^2 score
Building #18	0.7033 → 0.7069
Building #22	0.6983 → 0.7376
Building #25	0.7802 → 0.8058
Building #78	0.8141 → 0.8501

This table shows clear signs that even though the public holidays are marked as occupied (maybe all weekdays are set to be occupied by default for some buildings), setting them to be unoccupied rises the precision of the predictions since the buildings were probably unoccupied or operated with just limited resources. As a result, this enhancement stays employed.

Adding average features

Three extra features are added and their effect on the model's performance is evaluated. The first added feature is 'meas_month_avg' which is calculated as an average value of input power for each month. The second newly included feature is an average temperature in a month called 'temp_month_avg'. The last added feature is 'meas_dayofweek_avg' that is obtained as an average value of input power taking into consideration only the same days of the week in a month, e.g. the value for 'meas_dayofweek_avg' feature for each Monday in June 2022 would be determined as an average value of all Mondays in June 2022. This last feature should target the different heating approaches on each day of the week.

Based on the evaluation, the 'temp_month_avg' feature brings basically no improvement so it can be omitted. On the contrary, both features 'meas_month_avg' and 'meas_dayofweek_avg' showed slight improvements; however, they can be used only for analysis of the data not for future predictions because these values would not be available in the future as they are derived only for historical and current measurements.

Adding lag features

For future predictions in time series, lag features can be useful. The idea behind lag features is to use the value of a variable at previous time steps as a feature for predicting the value of the variable at the current time step. Lag features that reflect the value of the input power a week ago ('lag_1week') and a year ago ('lag_1year') are implemented. The lag features do not outperform the average features in data analysis, but they do not harm the performance.

Relabeling

Even though gradient boosting algorithms are supposed to be resilient to the type of values of the features, in this experiment, zero values in features are replaced. The 'dayofweek' feature range is changed from 0-6 to 1-7 and *False* values in 'occ' feature (occupancy of the building) are mapped to -1 instead of 0. Zero values might cause some problems in certain algorithms; however, in this case, the accuracy remained the same. Thus, the range 1-7 from the 'dayofweek' feature is kept as it makes sense and the occupancy values are changed back to 0 and 1.

Changing the split size between train and test sets

When performing the split into the training and the test datasets, there is a function `train_test_split()` from the scikit-learn library that does this splitting operation and has a parameter called `test_size`. This parameter specifies the percentage of the initial data that is allocated to the test set. The alteration between 20%, 25%, and 30% is examined. The effect on the model's performance varies from building to building so there is no clear answer to which percentage works the best, but it seems that the 30% fraction (`test_size=0.3`) yields reasonable results.

Altering the subsample fraction

Another important parameter in the *XGBRegressor* is the value of the *subsample*. In general, it's a good practice to set the subsample parameter to a value between 0.5 and 0.8 since this will balance the trade-off between overfitting and underfitting. Values of subsample fractions equal to 0.5, 0.6, 0.7, 0.8, and 1 are tested. Again, the accuracy shows no major changes and the behavior depends on the dataset itself. Nevertheless, the values of 0.6 and 0.7 resulted in the best outputs in most of the cases; hence, the average of 0.65 is set as the final value of the *subsample* parameter.

Choosing a suitable loss function

The influence of the squared error (L_2) and the Huber loss function are inspected. Results of R^2 scores for both loss functions can be seen in Table 4.5.

The Huber loss function is more robust to outliers than the squared error loss function, but since the data has already been treated for outliers, changing

Feature Importance

XGBoost library has a function `plot_importance()` that enables plotting how important certain features are in building the ensemble, see Fig. 4.15. The hierarchy of the importance of features is different for each of the buildings, but the dominant features keep repeating.

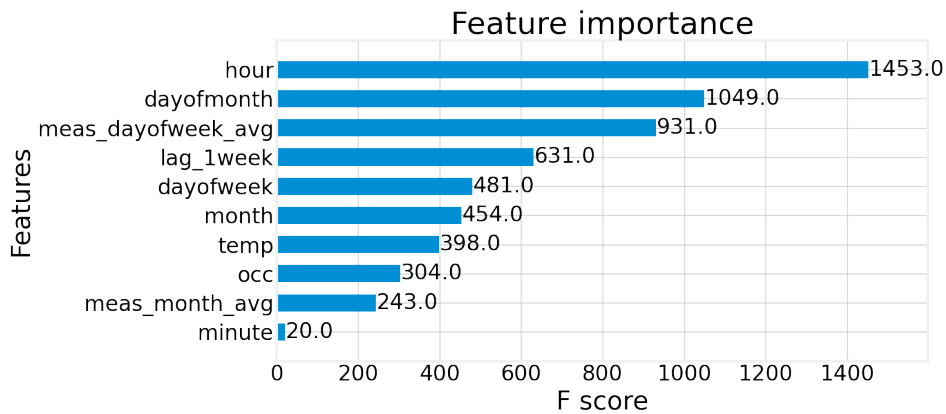


Figure 4.15: Feature importance plot for building No. 22

Visualization of decision trees

XGBoost library is also equipped with `plot_tree()` function which allows plotting any decision tree in the sequence. The first decision tree in the ensemble for the dataset of the first building building is depicted in Fig. 4.16.

In the next chapter, the final XGBoost model is compared to the scikit-learn and TOWT models.

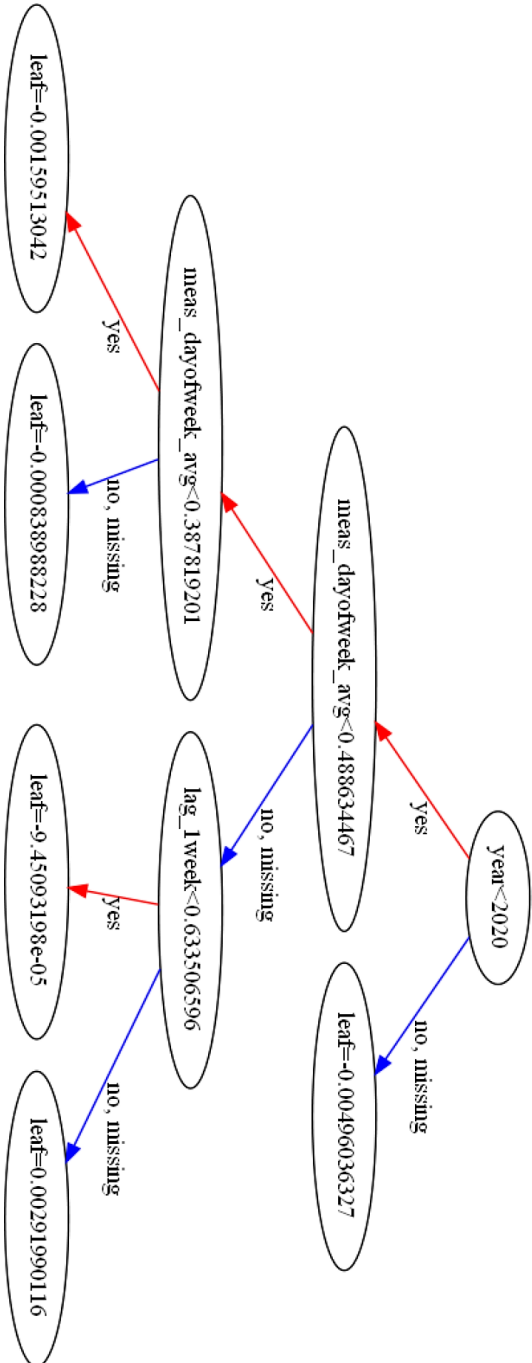


Figure 4.16: The first decision tree in the ensemble for building No. 0

Chapter 5

Results

This chapter is devoted to the comparison of predictive models with the aim of finding the one that performs the best on the given data for 124 smart buildings in Prague. First, two gradient boosting models that are developed as a practical part of this master thesis are compared. Then, the better of those two is set against the TOWT model that is being used in Energocentrum plus s.r.o. company for practical purposes.

Following statistical metrics for regression are used to compare the performance of models [35] [22]:

- *Mean*
- *STD (Standard Deviation)*
- *MSE (Mean Squared Error)* - measures the average squared difference between the predicted values and the real values
 - smaller values signify more precise estimates
- R^2 - determines the proportion of the variance in the target variable that is explained by the model
 - reaches values between (0,1)
 - the higher the value, the better the model is in making predictions
- *CV(RMSE) (Coefficient of Variation of the Root-Mean Squared Error)* - represents the RMSE as a percentage of the mean of the target variable
 - the value of 0 would indicate that the model's predictions are equal to the true target values

5.1 Comparison of the Scikit-learn and the XGBoost Models

As discussed in the previous chapter, two gradient boosting models are developed, one using the *GradientBoostingRegressor* class from the scikit-learn library (further referred to as SKLEARN model) and the other one using the *XGBRegressor* class from the XGBoost library (further referred to as XGB model). All the statistical metrics and comparison figures can be obtained by running the script *comparison_xgb_sklearn.py*.

The accuracy of the models is compared using two common statistical measures, namely the R-squared value (R^2) and the mean squared error (MSE). The buildings that reach R^2 values outside of the (0,1) are dropped from the evaluation. Fig. 5.1 demonstrates histograms of R^2 scores and Fig. 5.2 captures histograms of MSE values. It is noticeable that the XGB model reaches slightly higher values of R^2 score and lower values of MSE than the SKLEARN model which is desired. The SKLEARN model is able to get slightly better predictions for only 30 buildings out of the 124 buildings.

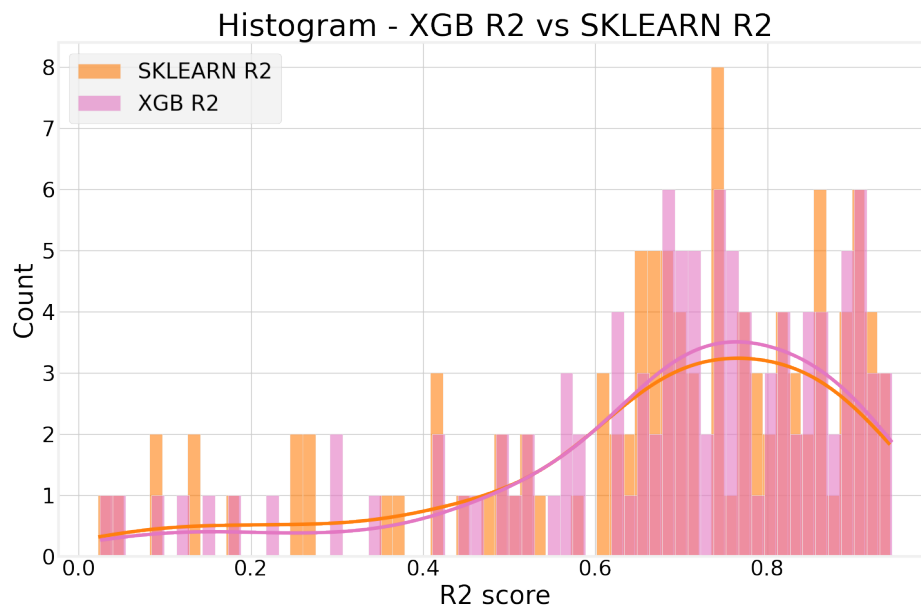


Figure 5.1: R^2 scores histogram - Comparison of the SKLEARN model against the XGB model

In the next step, 15 buildings with the highest differences in R^2 scores and MSE values between the two models are explored. These buildings' evaluation metrics are illustrated in Fig. 5.3 and Fig. 5.4. Except for the 2 buildings with indices 51 and 76, the XGB model is always more accurate. In addition, building No. 76 has many missing values and building No. 51 has a weird

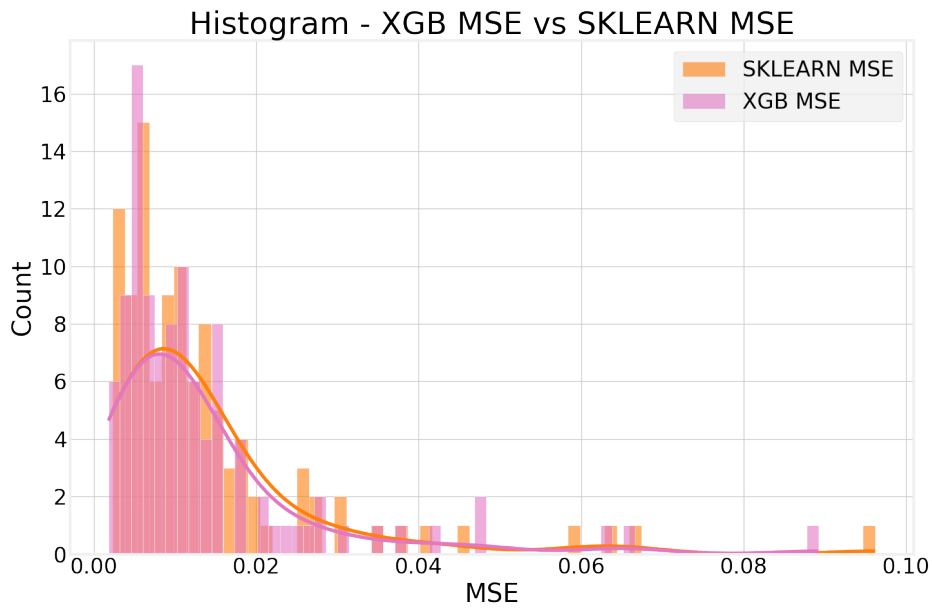


Figure 5.2: MSE histogram - Comparison of the SKLEARN model against the XGB model

drop in the input power values in the test set, so it is difficult to train the models on such datasets properly.

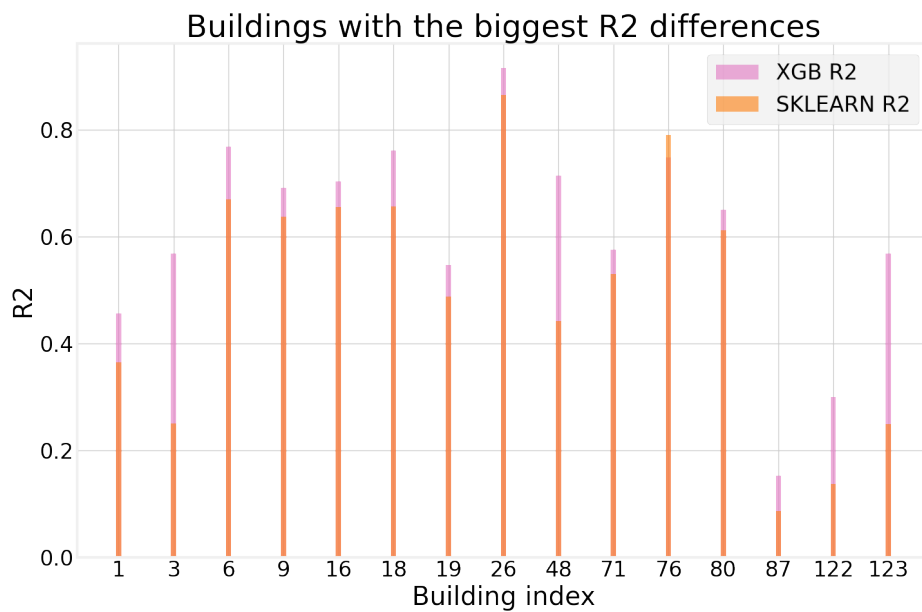


Figure 5.3: The biggest differences in R^2 scores of the SKLEARN and XGB models

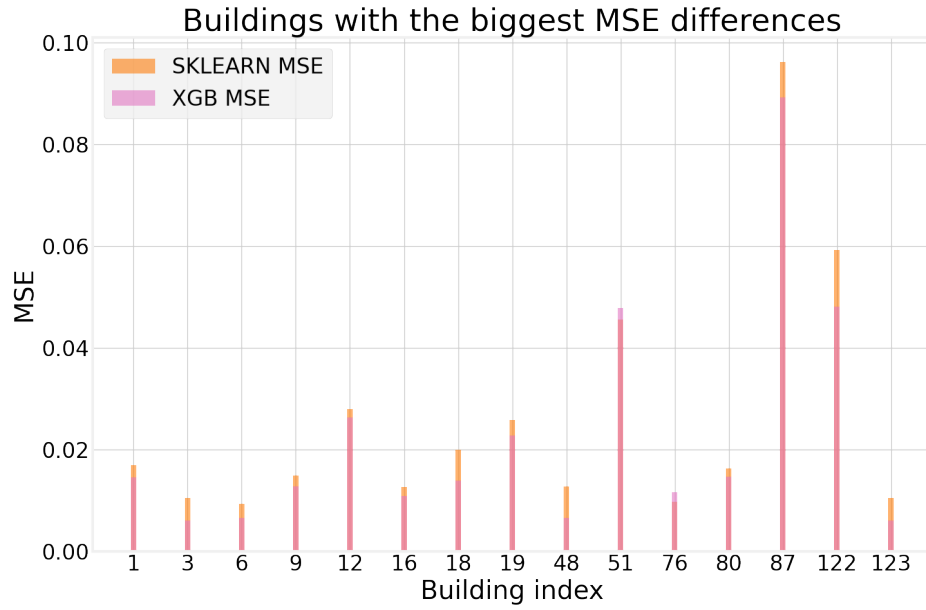


Figure 5.4: The biggest dissimilarities in MSE of the SKLEARN and XGB models

The final performance of both models is summarized in Table 5.1, these metrics are calculated as averages across all the buildings in the dataset. As conclusion, it is shown that the XGB model performs a bit better and thus, is chosen as the winner. In the next section, the XGB model is compared to the TOWT model.

Table 5.1: Evaluation of the overall models' performance

Model	Avg R^2 score	Avg MSE
SKLEARN model	0.6692	0.0141
XGB model	0.6879	0.0135

5.2 Comparison of the XGBoost and the TOWT Models

TOWT stands for Time-of-Week-and-Temperature model and is a type of time series forecasting model (piecewise linear model) that is used to predict the values of a variable based on the time of the week and temperature. This model is commonly used in predicting energy consumption, it takes into account the daily and weekly patterns in the data, as well as the impact of temperature on energy consumption. [12]

The script `comparison_xgb_towt.py` enables to plot the comparison graphs between the XGB and TOWT models based on the mean, STD, R^2 , MSE, and CV(RMSE) statistical metrics. The plots reflecting the actual values of the target variable (the input power) against the predictions made by both models can be acquired by running the script `xgb_towt.py`.

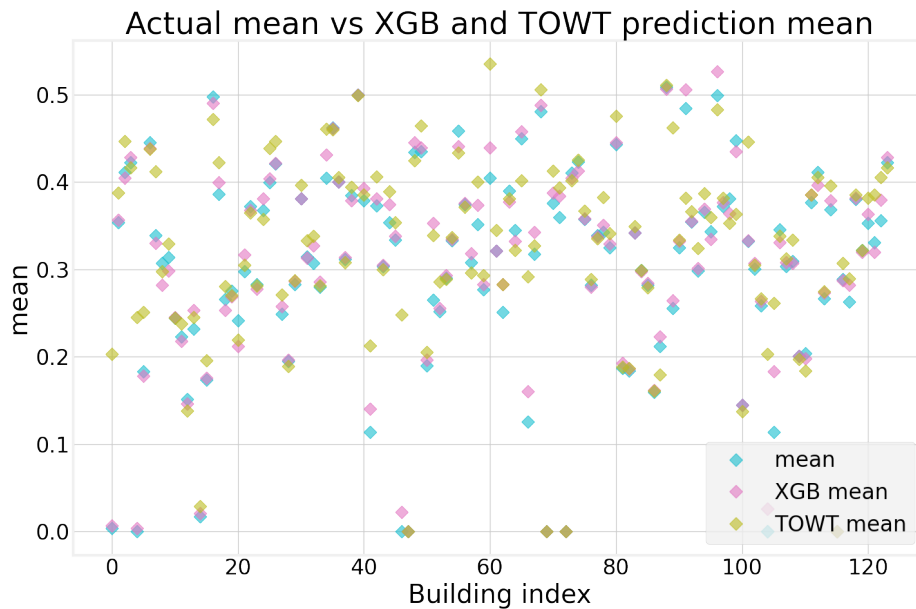


Figure 5.5: Comparison of the true mean values against the mean values of TOWT and XGB models

Firstly, the mean and standard deviation values are examined (see Fig. 5.5 and 5.6). The blue points denote the actual mean and STD values of the provided input datasets, the pink points signify the XGB model's values, and the yellow points depict the values of the TOWT model. The XGB model mimics the actual values more closely because the pink points are, in general, closer to the blue points (real values) than the yellow points (TOWT). To make it more clear that the XGB model performs slightly better, box plots of

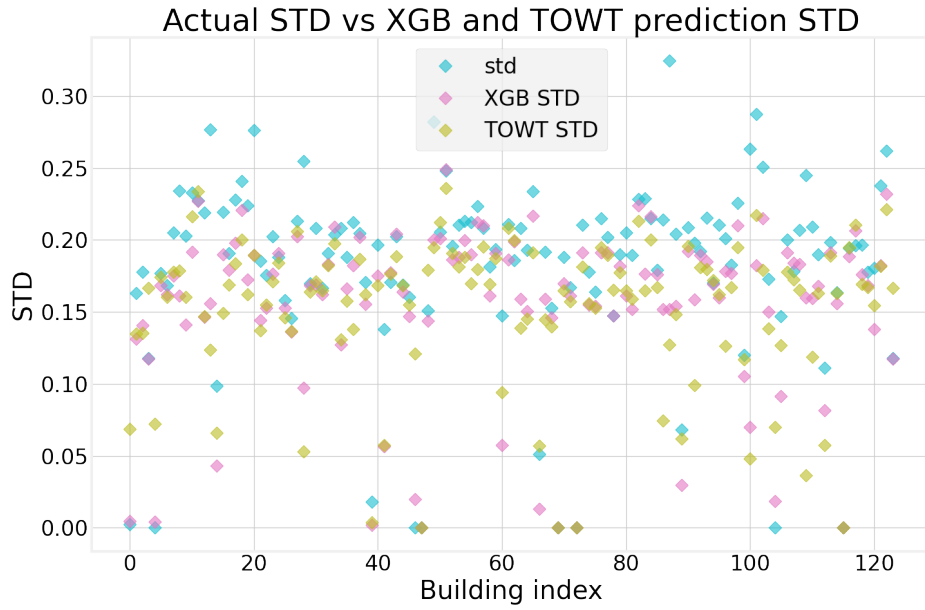


Figure 5.6: Comparison of the actual STD against the STD values of TOWT and XGB models

Actual mean vs XGB and TOWT prediction mean

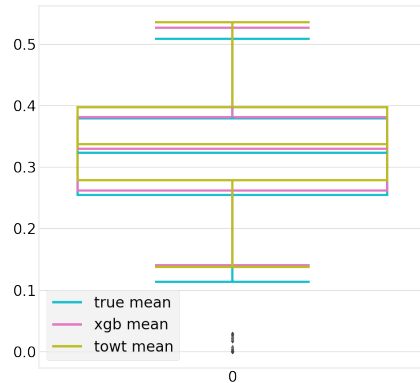


Figure 5.7: Boxplots of the actual, XGB, and TOWT mean values

Actual STD vs XGB and TOWT prediction STD

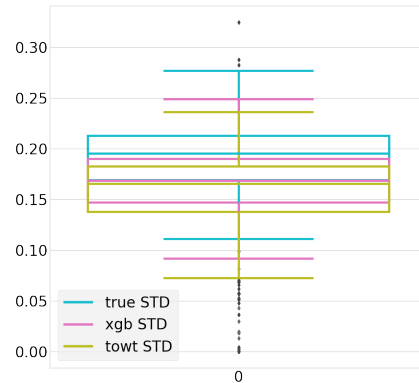


Figure 5.8: Boxplots of the actual, XGB, and TOWT STD values

mean and STD values can be observed in Fig. 5.7 and Fig. 5.8, respectively. The same color coding is used in the box plots.

Further, the performance of the XGB model and the TOWT model is evaluated using typical statistical measures such as the R^2 score, MSE, and CV(RMSE). Once more, the buildings that reach R^2 values outside of the (0,1) are omitted from the evaluation. Histograms of R^2 scores are illustrated in Fig. 5.9, histograms of MSE values are depicted in Fig. 5.10, and histograms of CV(RMSE) values are demonstrated in Fig. 5.11. It is easily observable that

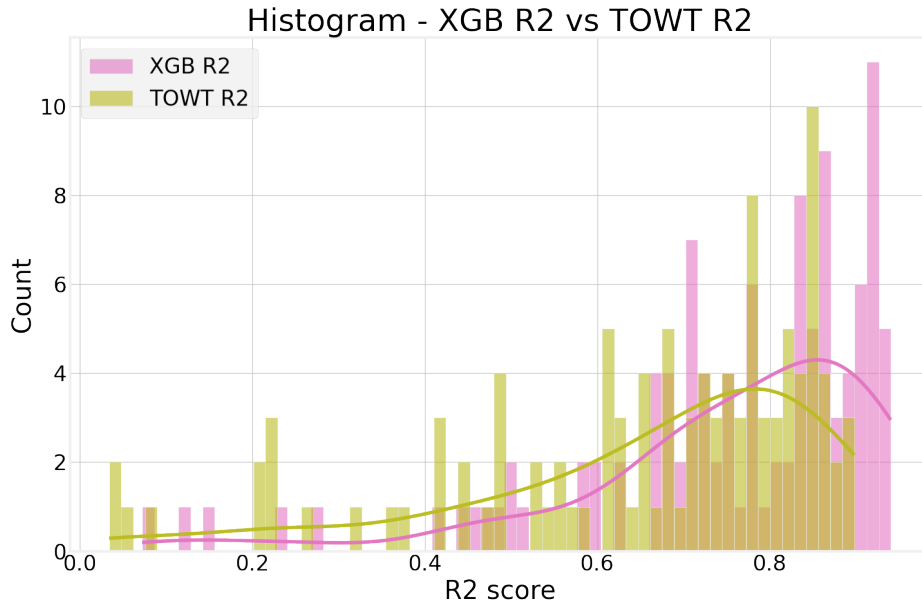


Figure 5.9: R^2 scores histogram - Comparison of the TOWT model against the XGB model

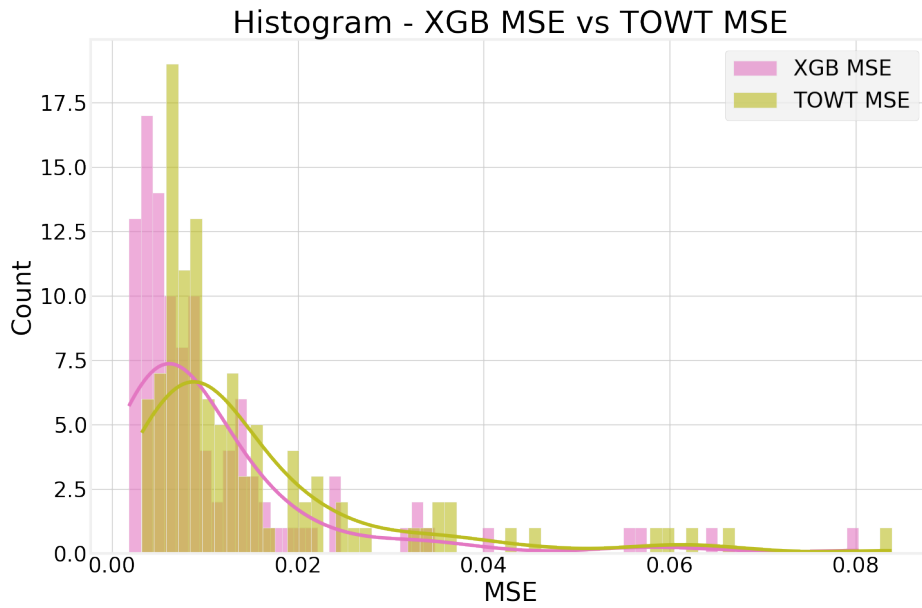


Figure 5.10: MSE histogram - Comparison of the TOWT model against the XGB model

the XGB model outperforms the TOWT model, there are only 3 buildings (with indices 8, 9, and 40) where the TOWT model yields more precise predictions. Buildings No. 8 and 9 have data recorded only for one year

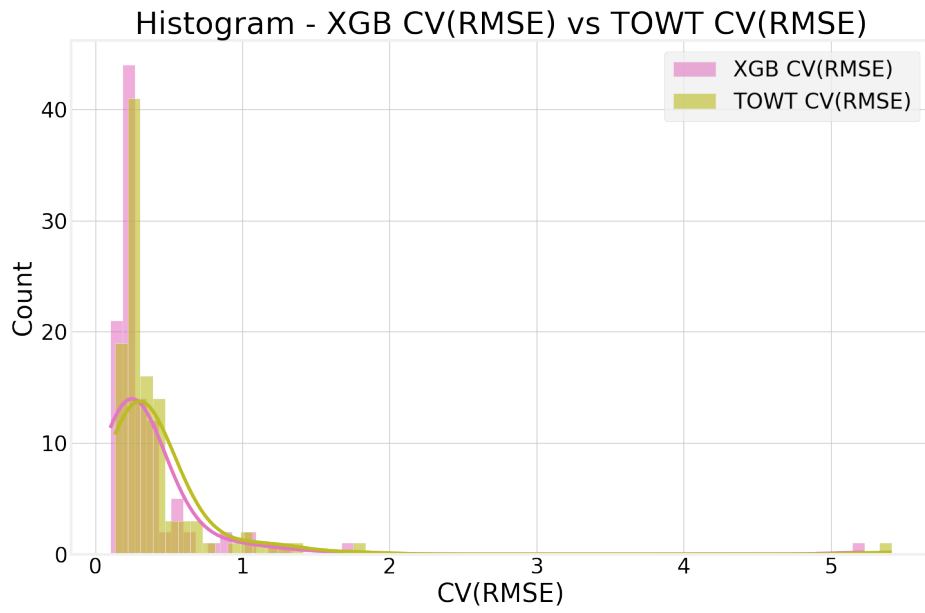


Figure 5.11: CV(RMSE) histogram - Comparison of the TOWT model against the XGB model

which might not be sufficient for the XGB model to result in reasonable outcomes.

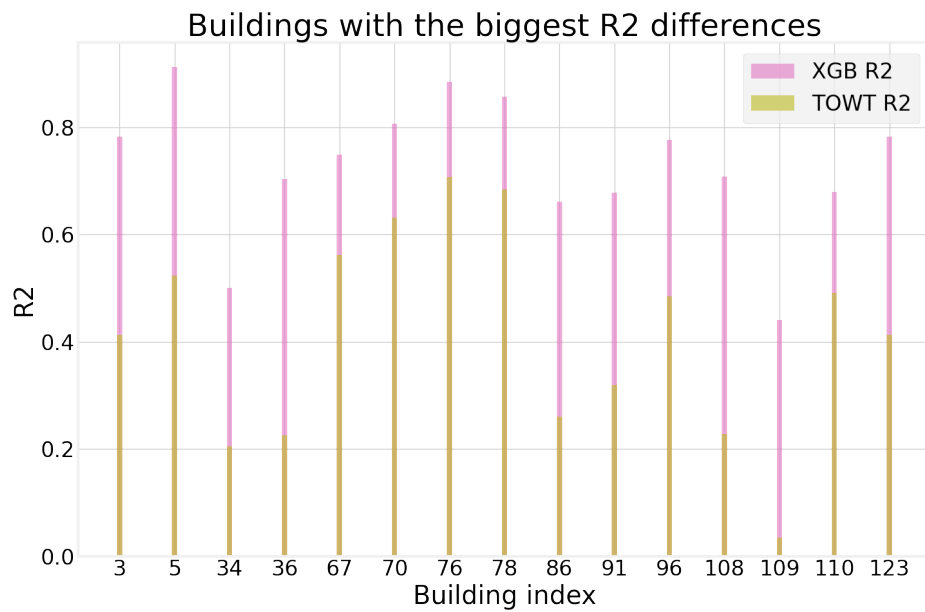


Figure 5.12: The biggest differences in R^2 scores between the TOWT and XGB models

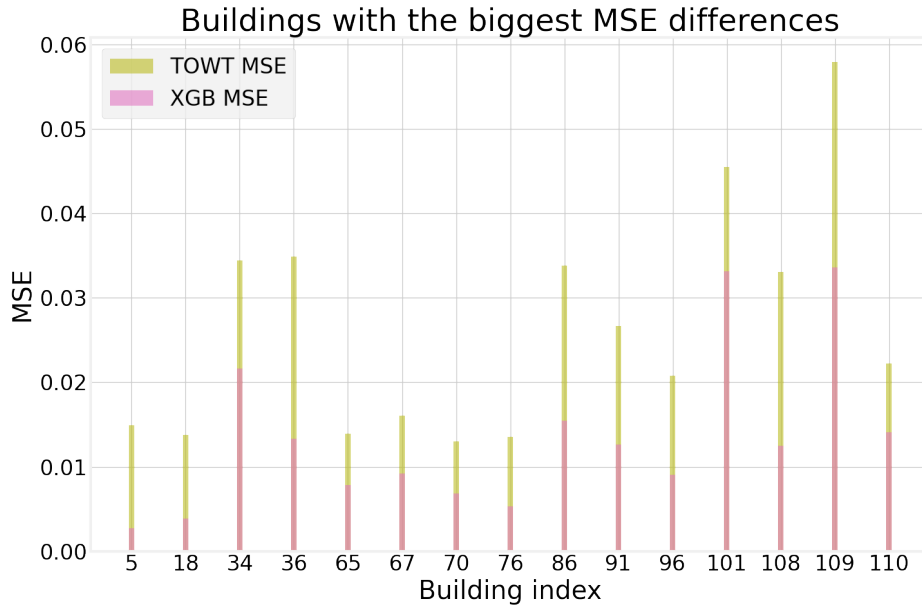


Figure 5.13: The biggest dissimilarities in MSE values between the TOWT and XGB models

Again, 15 buildings with the biggest differences of R^2 scores and MSE values between the two models are detected and their evaluation metrics are depicted in Fig. 5.12 and Fig. 5.13. For all of those buildings, the XGB model resulted in more accurate predictions.

The biggest difference in the R^2 score between the XGB and TOWT models is found for building No. 108. The whole prediction period is shown in Fig. 5.14, where it can be observed that the TOWT model is not able to discover the lower input power values at the beginning of the prediction period, while the XGB model is able to adapt and follows the trend of the actual target values much more closely. It seems that the TOWT model sticks to a quite constant prediction pattern. When it is zoomed to the first week of predictions starting from Monday for building No. 108, it is even more visible that the TOWT model does not do a good job for this dataset (see Fig 5.15). On the other hand, the predictions that can be seen in Fig. 5.16 look to be very appropriate for both of the models, even though the XGB model again yields slightly more precise estimations.

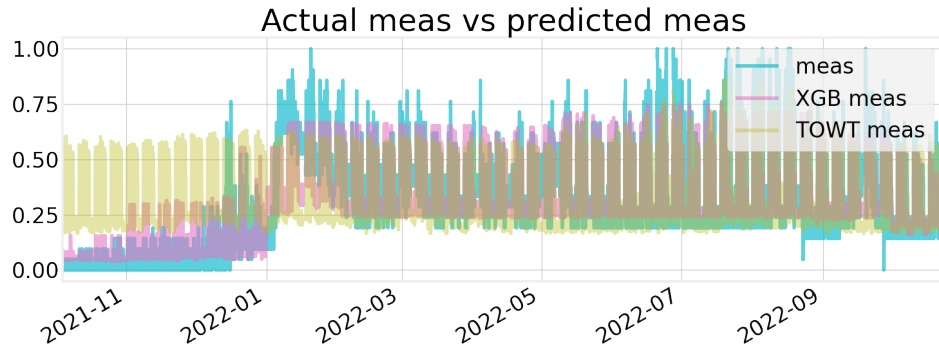


Figure 5.14: Predictions of input power for building No. 108

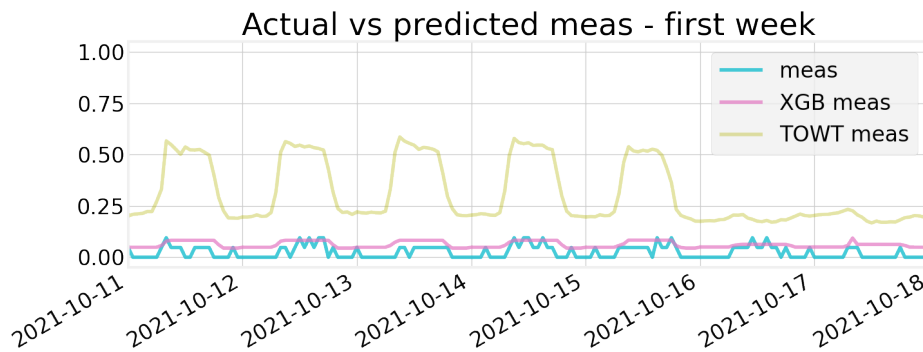


Figure 5.15: First week of predictions for building No. 108

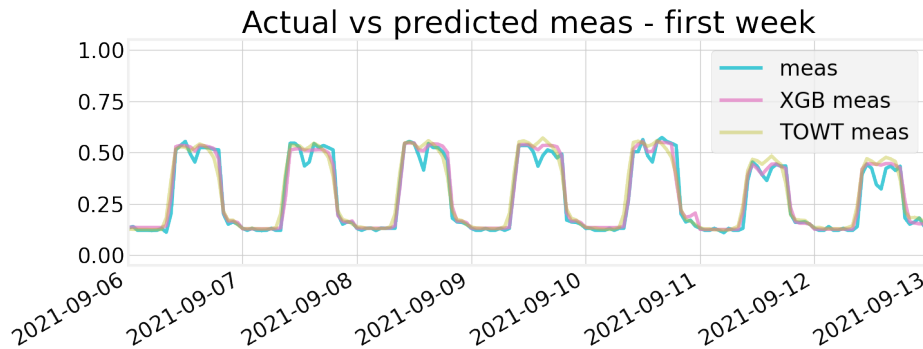


Figure 5.16: First week of predictions for building No. 29

Table 5.2 summarizes the average evaluation metrics for both models. Recall that lower MSE values are desired, while higher R^2 scores show more precise predictions.

Taking into consideration the TOWT model’s low complexity and computational cost, it shows reasonable predictions, but it is not near the precision of the XGB model. The input features of the TOWT model are only tem-

Table 5.2: Evaluation of the overall models' performance

Model	Avg R^2 score	Avg MSE	Avg CV(RMSE)
TOWT model	0.6548	0.0151	0.4359
XGB model	0.7517	0.0113	0.3753

perature and timestamp so the resulting performance of the model is highly dependent on the quality of the input data and the ability of the model to capture the underlying patterns in the data. On the contrary, the XGB model is provided with additional "datetime" and "average" features and is, generally, more flexible and more robust to noisy data since it is an ensemble of learners.

To sum up, based on Table 5.2 there is no doubt about which model performs better. The XGB model's accuracy is 10% higher than the accuracy of the TOWT model. And both the average mean squared error and the coefficient of variation of the root-mean squared error are lower for the XGB model which means that the XGB model makes predictions that are closer to the target values. The error metrics are always desired to be as close to zero as possible.

So to conclude the results chapter, the XGB model achieves the best performance and thus, is a clear winner among the 3 models that are here compared.



Conclusion

The content of this thesis begins with an introduction to machine learning followed by an explanation of ensemble methods with a primary focus on boosting techniques. Then, the Gradient Boosting Machine algorithm is explained in detail. This theoretical part fulfills the first point of the assignment.

The major goal was to develop a gradient boosting model in Python that is able to make accurate predictions of energy consumption which was achieved and that covers the second and third points of the given assignment. Precise energy consumption predictions can help companies to manage their energy supply and improve the efficiency of energy distribution. By having the energy consumption estimates, it can be ensured that there is enough capacity to the demand and any blackouts can be avoided. Further, energy efficiency can be improved by identifying areas where energy consumption is high and optimizing the energy losses. Finally, accurate energy consumption predictions can be also used to integrate renewable energy sources and to help support consumers in managing their energy usage.

The final step of the assignment is satisfied by validating the developed gradient boosting model on data supplied by Energocentrum plus s.r.o. From the experiments conducted in this work, it can be concluded that both gradient boosting methods outperformed the TOWT model in terms of the accuracy of predictions. The XGBoost model reached the lowest prediction errors of all three compared models. In general, the XGBoost library is a powerful and flexible library for gradient boosting, it's efficient in handling large datasets and high-dimensional data, and it is widely used in many applications as it supports parallel processing and provides regularization techniques that can prevent the model from overfitting.

The performance of the final XGBoost model yields promising results, but it is dependent on the quality of the input data. Many preprocessing techniques are employed to increase the quality of the data fed directly to the XGBoost model. Of course, there is certainly room for further modifications and enhancements. One suggestion would be to train the model so that it is able to recognize whether the building uses more energy for heating or for AC because there is a difference between the two. The heat gets accumulated in

the walls of buildings, while air conditioning happens in an instant matter which results in non-identical energy consumption trends in summer and in winter.

However, the current model achieves good accuracies of energy estimates and proves to better track more irregular energy consumption trends than the TOWT model which is much simpler.



Bibliography

- [1] A. Syed. Ai vs. ml vs. dl (vs. nn). [Online]. Available: <https://medium.com/a-coders-guide-to-ai/ai-vs-ml-vs-dl-vs-nn-f6968db769d1>
- [2] P. Domingos, “A few useful things to know about machine learning,” *communications of the acm*, 2012. [Online]. Available: <https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf>
- [3] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with applications in R*, 2nd ed. Springer New York, NY, 2013. [Online]. Available: https://hastie.su.domains/ISLR2/ISLRv2_website.pdf
- [4] C. Team. Ensemble methods. [Online]. Available: <https://corporatefinanceinstitute.com/resources/data-science/ensemble-methods/>
- [5] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, <http://www.deeplearningbook.org>.
- [6] Xgboost. [Online]. Available: <https://www.geeksforgeeks.org/xgboost/>
- [7] A. Pal. Gradient boosting trees for classification: A beginner’s guide. [Online]. Available: <https://affine.ai/gradient-boosting-trees-for-classification-a-beginners-guide/>
- [8] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.
- [9] A. Natekin and A. Knoll, “Gradient boosting machines, a tutorial,” *Frontiers in Neurorobotics*, vol. 7, 2013. [Online]. Available: <https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021>
- [10] S. E.Yu. and P. E.A., “Smart buildings energy savings with gradient boosting algorithm,” 2018. [Online]. Available: <https://ceur-ws.org/Vol-2267/318-322-paper-60.pdf>
- [11] L. A. S. Eugeny Yu. Shchetinin, Vladimir S. Melezhik, “Improving the energy efficiency of the smart buildings with the boosting

- algorithms,” *Selected Papers of the 12th International Workshop on Applied Problems in Theory of Probabilities and Mathematical Statistics (Summer Session) in the framework of the Conference “Information and Telecommunication Technologies and Mathematical Modeling of High-Tech Systems”, Lisbon, Portugal, October 22–27, 2018*, 2018. [Online]. Available: <https://ceur-ws.org/Vol-2332/paper-08-014.pdf>
- [12] S. Touzani, J. Granderson, and S. Fernandes, “Gradient boosting machine for modeling the energy consumption of commercial buildings,” *Energy and Buildings*, vol. 158, pp. 1533–1543, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0378778817320844>
- [13] H. Lu, F. Cheng, X. Ma, and G. Hu, “Short-term prediction of building energy consumption employing an improved extreme gradient boosting model: A case study of an intake tower,” *Energy*, vol. 203, p. 117756, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S036054422030863X>
- [14] P. Lillian Pierson. Python for data science essential training part 2. [Online]. Available: <https://www.linkedin.com/learning/python-for-data-science-essential-training-part-2/machine-learning-rocks?autoplay=true&u=3514>
- [15] A. Odubela. Supervised learning essential training. [Online]. Available: <https://www.linkedin.com/learning/supervised-learning-essential-training/supervised-machine-learning-and-the-technology-boom?autoplay=true&u=3514>
- [16] K. McCormick. Advanced predictive modeling: Mastering ensembles and metamodeling. [Online]. Available: <https://www.linkedin.com/learning/advanced-predictive-modeling-mastering-ensembles-and-metamodeling/the-most-accurate-machine-learning-models?autoplay=true&u=3514>
- [17] H. Patel. What is feature engineering — importance, tools and techniques for machine learning. [Online]. Available: <https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b026>
- [18] F. Nwanganga. Machine learning with python: Decision trees. [Online]. Available: [linkedin.com/learning/machine-learning-with-python-decision-trees/making-decisions-with-python?autoplay=true&u=3514](https://www.linkedin.com/learning/machine-learning-with-python-decision-trees/making-decisions-with-python?autoplay=true&u=3514)
- [19] T. M. Hehn, J. F. P. Kooij, and F. A. Hamprecht, “End-to-end learning of decision trees and forests,” *International Journal of Computer Vision*, 2020. [Online]. Available: <https://doi.org/10.1007/s11263-019-01237-6>
- [20] S. Hara and K. Hayashi, “Making tree ensembles interpretable,” 2016. [Online]. Available: <https://arxiv.org/abs/1606.05390>

- [21] F. Divina, M. García Torres, F. A. Gómez Vela, and J. L. Vázquez Noguera, “A comparative study of time series forecasting methods for short term electric energy consumption prediction in smart buildings,” *Energies*, vol. 12, no. 10, 2019. [Online]. Available: <https://www.mdpi.com/1996-1073/12/10/1934>
- [22] X. Liu, H. Tang, Y. Ding, and D. Yan, “Investigating the performance of machine learning models combined with different feature selection methods to estimate the energy consumption of buildings,” *Energy and Buildings*, vol. 273, p. 112408, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0378778822005795>
- [23] C. Miller, L. Hao, and C. Fu, “Gradient boosting machines and careful pre-processing work best: ASHRAE great energy predictor III lessons learned,” *CoRR*, vol. abs/2202.02898, 2022. [Online]. Available: <https://arxiv.org/abs/2202.02898>
- [24] pickle — python object serialization. [Online]. Available: <https://docs.python.org/3/library/pickle.html>
- [25] P. Lillian Pierson. Python for data science essential training part 1. [Online]. Available: <https://www.linkedin.com/learning/python-for-data-science-essential-training-part-1/data-science-life-hacks?autoplay=true&u=3514>
- [26] D. Yadav. Categorical encoding using label-encoding and one-hot-encoder. [Online]. Available: <https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd>
- [27] F. Nwanganga. Machine learning with python: Foundations. [Online]. Available: <https://www.linkedin.com/learning/machine-learning-with-python-foundations/machine-learning-in-our-world?autoplay=true&u=3514>
- [28] M. Sharma. Explained: ML transformation scaling. [Online]. Available: <https://towardsdatascience.com/transformation-scaling-of-numeric-features-intuition-7f4436e8e074>
- [29] A. McDonald. Seaborn heatmap for visualising data correlations. [Online]. Available: <https://towardsdatascience.com/seaborn-heatmap-for-visualising-data-correlations-66cbef09c1fe>
- [30] ——. Seaborn pairplot: Enhance your data understanding with a single plot. [Online]. Available: <https://towardsdatascience.com/seaborn-pairplot-enhance-your-data-understanding-with-a-single-plot-bf2f44524b22>
- [31] Xgboost documentation. [Online]. Available: <https://xgboost.readthedocs.io/en/stable/>

- [32] `sklearn.ensemble.gradientboostingregressor`. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor>
- [33] N. Kumar. Difference between gbm (gradient boosting machine) and xgboost (extreme gradient boosting). [Online]. Available: <http://theprofessionalspoint.blogspot.com/2019/02/difference-between-gbm-gradient.html>
- [34] A. Anghel, N. Papandreou, T. P. Parnell, A. D. Palma, and H. Pozidis, “Benchmarking and optimization of gradient boosted decision tree algorithms,” *CoRR*, vol. abs/1809.04559, 2018. [Online]. Available: <http://arxiv.org/abs/1809.04559>
- [35] A. González-Vidal, A. P. Ramallo-González, F. Terroso-Sáenz, and A. Skarmeta, “Data driven modeling for energy consumption prediction in smart buildings,” in *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 4562–4569.