# Improving Elastic Gradient Boosting: Catching Concept Drift in Smart Meter Time Series Data

By

Sebastian Štros

Submitted to

Central European University

Department of Network and Data Science

*In partial fulfillment of the requirements for the degree of*

Master of Science in Social Data Science

Supervisor: DSc Marton Karsai

External Supervisor: MSc Tomáš Šatura

Vienna, Austria

May 2025

# AUTHOR'S DECLARATION

I, the undersigned, **Sebastian Štros**, candidate for the MSc degree in Social Data Science declare herewith that the present thesis titled "Improving Elastic Gradient Boosting: Catching Concept Drift in Smart Meter Time Series Data" is exclusively my own work, based on my research and only such external information as properly credited in notes and bibliography. I declare that no unidentified and illegitimate use was made of the work of others, and no part of the thesis infringes on any person's or institution's copyright.

I also declare that no part of the thesis has been submitted in this form to any other institution of higher education for an academic degree.

Vienna, 22 May 2025

Sebastian Stros

# COPYRIGHT NOTICE



[1]



[1] Icon by Creative Commons

# Abstract

This thesis improves on the elastic Gradient Boosting Decision Tree algorithm (eGBDT) by limiting its memory use and increasing its accuracy. eGBDT is a successful incremental learning algorithm. Because it deletes weak learners which are outdated, it maintains high accuracy on conceptually drifted data. This thesis offers two versions which improve on eGBDT's accuracy and memory use. The first version contribution-based eGBDT (cbeGBDT) improves eGBDT by making weak learner deletion dependent only on performance and not on index as in the original eGBDT. The second version, Accuracy Updated eGBDT (AUeGBDT) adds to cbeGBDT weighting mechanism inspired by the Accuracy Updated Ensemble. Improving eGBDT accuracy and memory use is motivated by the new and important application of incremental learning on smart meter time series data. Smart meter data need to be accurately forecasted to maintain energy grid stability in the times of the massive renewable energy sources roll out and transport electrification. On this regression task, the cbeGBD outperforms the original eGBDT, non-retraining GBDT, but is comparable to a non-icrementally learning GBDT model which retrains on every batch.

# Acknowledgments

Firstly, I would like to thank Tomáš Šatura for hiring me into the intellectually inspiring company Mycroft Mind and guiding me towards this research project. By extension, I would like to thank my colleagues at Mycroft Mind for critiquing my presentation. Special thanks goes to Adam Pešl with whom I discussed my ideas most frequently and who explained to me a number of time series concepts. I would also like to thank Quido Haškovec for providing feedback on my academic writing. I would like thank my supervisor Marton Karsai for his feedback on my thesis drafts. Last but not least, I would like to thank my parents and my friends for their undying support.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

There is a growing need to predict the production and consumption of energy by limited memory devices such as smart meters to maintain the stability of the electric grid. Thanks to the much-needed rise of renewables, our energy mixes are more environmentally sustainable; however, they create new pressures on electric grid stability [2, 3, 4]. Most households in the most developed nations have smart meters installed. Until now, these simple devices were only sending data to some data center where crude aggregate forecasts on energy consumption were made [5]. To improve grid stability, scientists are experimenting with the deployment of forecasting software into the edges of the grid itself such as into smart meters [5]. Accurate forecasts could substantially improve grid stability. This thesis aims to further the development of such software by proposing a new low-memory, low-compute algorithm.

Since smart meters are highly limited in both memory and computational power, their predictive algorithms need to learn *incrementally* (continually) from incoming data while forgetting old data. Smart meters cannot store data from the previous years due to memory constraints. In addition, energy consumption patterns shift when consumers adopt new energy-consuming or producing appliance such as kettle, vacuum cleaner or a solar panel; external factors such as seasons, holidays, or climate change further alter demand patterns. These shifts, known as *concept drift*, make accurate forecasting challenging. Concept drift, as defined in Concept Drift 2.1, refers to the degradation of a machine learning model's predictive performance due to discrepancies between the training data and the data encountered during deployment. Effective incremental learning would address the issue of concept drift by continuously adapting to changing characteristics of the data. Such incremental learning is not applicable only in edge computing but in many other areas where concept drift occurs.

Gradient boosting decision tree algorithm (GBDT) is a good starting point for building incremental learning extensions for edge computing due to its relatively high predictive accuracy combined with low computational and memory requirements (compared to neural networks and

other complex models). Moreover, GBDT does not rely on a specific type of weak learner such as Hoeffding tree. That makes GBDT versatile in its application as any type of weak learner can be chosen for a specific task.[1] Its elastic version (eGBDT) learns incrementally and in two published paper it has outperformed other incremental learning approaches on most tested datasets [6, 7]. However, so far these were classification tasks and allowed eGBDT's memory use to be potentially infinite. The current discussion prompts the research question:

*How can the Elastic GBDT algorithm be adapted for regression tasks to limit memory use and computational resources while enhancing predictive accuracy, and ensuring compatibility across various types of decision trees?*

We will now present in a very abbreviated form the original elastic GBDT as proposed by Wang et al. and our improvements to it [6]. The full explanation of eGBDT is in Section 2.2.3 while our improvements are in Section 3.1. Elastic GBDT is used in a context where it continually receives data-batches and makes prediction based on the latest data-batch for the next data-batch. This is called *batch learning*. The main innovation of the elastic GBDT proposed by [6] is that it either deletes (prunes) all trees after index $m$ or retrains the whole ensemble depending on the value of $m$. If $m < M$, where $m$ is the index of the lowest error on the latest data batch and $M$ is some minimal size of the ensemble, the whole model is retrained. If $m > M$, only trees $m+1, m+2, ..., n$ are deleted (where $n$ is the last tree) and $L$ new so called 'incremental trees' are trained with indices $m+1, m+2, ..., m+L$. The new either fully retrained or incrementally trained model makes predictions for the next data-batch. Because of the dependency on indexing of trees and to differentiate from our version of eGBDT, I call this version index-based elastic GBDT.

We believe that this model can be improved because within the group of the pruned trees (trees after index $m$), contribution to the reduction of the prediction error can be highly heterogeneous. Perhaps, the whole error is created by a single tree while the remaining trees are reducing the error, but they do not reduce it enough to balance for the one conceptually outdated tree. The main research contribution of this thesis is to delete trees based on their individual contribution

[1] Base-learner and weak learner are used in this text interchangeably. Full explanation of Hoeffding tree and why it might be good to have incremental learning algorithm independent from the choice of weak learner is in Section 2

in reducing prediction error in wholly non-index-based manner. This *contribution-based elastic GBDT* (ebeGBDT) prunes only the trees whose contribution deteriorates with the arrival of batch. The more targeted pruning can save training costs by not deleting and training as many trees as the index-based elastic GBDT does. In addition, only as many trees as were deleted are trained. That keeps the number of trees and thus memory constant. We offer two different ways to determine how many trees should be pruned and retrained at the end of each batch. Furthermore, we propose a second algorithm where we combine the contribution-based pruning approach with weighting introduced by different incrementally learning algorithm named Accuracy Updated Ensemble (AUE first propsed Brezinsky et al. [8]). Weighting should improve accuracy even further, because it adapts on a more granular level than pruning and building new trees does.

The structure of the thesis is the following. The first section was the introduction. The second section is the literature review which presents what concept drift formally is and what are the incremental learning strategies we are building on. The third section presents the result of our work which cover the theoretical explanation of the two proposed algorithms, the setup of the experiment including the datasets, preprocessing steps, and the experimental results. The fourth section is the discussion of the limitations. The final fifth section is the conclusion.

# Chapter 2

# Literature Review and Methodology: Concept Drift and Incremental Learning

This chapter presents what concept drift is and what are the methods we are building on when proposing the two algorithms in the next section. Concept drift is explained here both formally and informally. The methods for incremental learning presented here are not exhaustive. While we the major approaches which gave rise to their own lineages and improvements are discussed, only the algorithms directly relevant to this research are explained in greater detail.

## 2.1 Concept Drift

The main challenge for accurately predicting energy data within smart meters is the issue of concept drift. Lu et al. define concept drift as "a phenomenon in which the statistical properties of a target domain change over time in an arbitrary way" [1]. Since smart meters cannot support large models or store extensive data, the concepts they have learned from training data change after and during deployment; thus, leading to less accurate predictions of the *target domain*. However, concept drift is not an issue peculiar to smart meters, but for energy data more generally. Reasons why the concept drift occurs in energy data are numerous: low quality and size of training data, qualitative difference between the context of training data and deployment data (e.g. training data come from less energy-consuming households whereas the model is deployed in an industrial setting), climate change slowly but surely changes weather patterns and therefore heating and energy patterns, households may arbitrarily change their energy consumption pattern by e.g. buying new appliance. There are many more reasons for concept drift occurring in energy data see various kinds of research working with concept drift in energy data: [9, 10, 11]. The listed examples served us to understand concept drift informally and to argue that concept drift occurs in smart meter energy data. However, concept drift has been

mathematically formalized. We will now present the formal definition of the three sources of concept drift which give rise to its four types. The formalization uses the language of probability distributions and it is elegantly illustrated by two figures borrowed from Lu et al. [1].

### 2.1.1 Sources of Concept Drift

The formal definition begins by the joint distribution $P(X, Y)$ [1, 12]. Where X are the independent variables and Y the dependent variables. The sources are visualized here: 2.1. For simplicity the visualization uses X as one independent variable, but there is no reason to limit the scope just to one. The $P(X, Y)$ distribution can be decomposed by the chain rule of probability into:

$$P(X, Y) = P(X) \cdot P(Y \mid X).$$

Hence, there can be three sources of concept drift which result from the joint distribution $P_t(X, Y)$ at time $t$ not being equal to $P_{t+\Delta}(X, Y)$ at time $t + \Delta$, for $\Delta > 0$. Why are there three possible sources and not a one more nor less? Because the decomposition leads to possible three combinations of changes over time:

1. **Source I (Virtual Drift):**

$$P_t(X) \neq P_{t+\Delta}(X) \quad \text{and} \quad P_t(Y \mid X) = P_{t+\Delta}(Y \mid X).$$

2. **Source II (Actual Drift):**

$$P_t(Y \mid X) \neq P_{t+\Delta}(Y \mid X) \quad \text{and} \quad P_t(X) = P_{t+\Delta}(X).$$

3. **Source III (Mixed Drift):**

$$P_t(X) \neq P_{t+\Delta}(X) \quad \text{and} \quad P_t(Y \mid X) \neq P_{t+\Delta}(Y \mid X).$$

for some $\Delta > 0$.

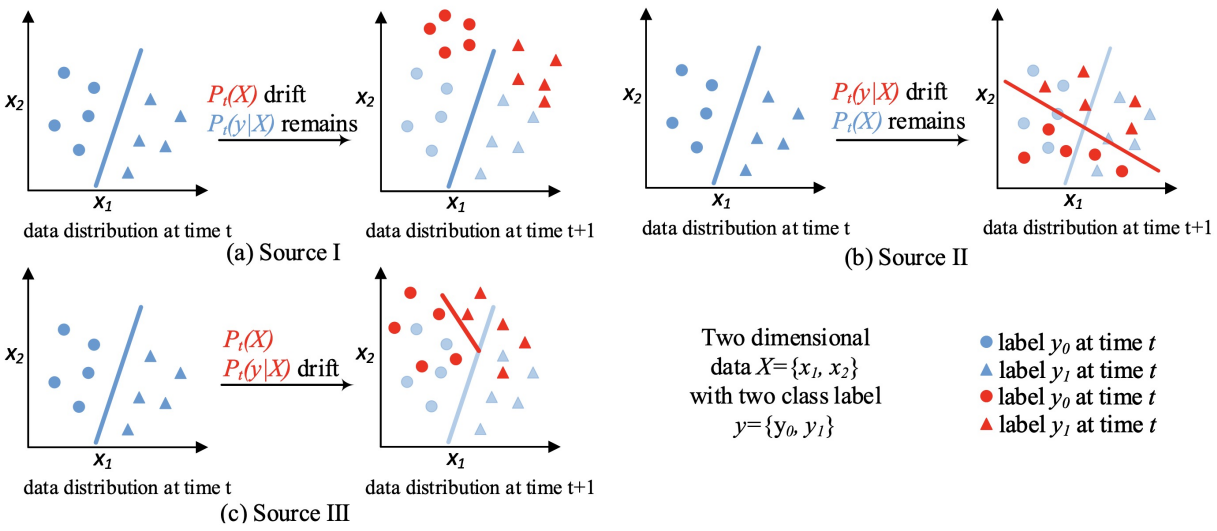


Figure 2.1: Concept Drift Sources. Adapted from Lu et al. (2018) [1]

The **Source I** is called Virtual Drift, because there is no *actual* change in concepts [13, 12, 1]. The concepts capturing the relationship between the X characteristics and the Y outcome variable are not changing. What is changing is the distribution of X and accordingly Y. This means that the production data are consistently e.g. of higher values than training data. Although the relationship between X and Y does not change, the model's performance is still likely to worsen, because the such higher values were hardly present in the training data. Hence it 'understands' them less than lower values.

The **Source II** is called Actual Drift, because the relationship between X and Y captured by $P(Y \mid X)$ after time $\Delta$ *actually* changed [1, 12]. This means that the concepts understood from the data at time $t$ no longer apply $t + \Delta$. Inevitably, the performance of the model will worsen. Because it does not capture the new concepts of $P_{t+\Delta}(Y \mid X)$.

The **Source III** is called Mixed Drift because it mixes Sources I and II [1].

### 2.1.2 Types of Concept Drift

The three sources of concept drift give rise to four recognized types of concept drift [1, 14]. These are formally described as: sudden, gradual, incremental and recurring ([12] make introduce detailed sub-types). Of course, these can be combined together. The types are well visualized in Concept Drift Types by Lu et al.: 2.2.

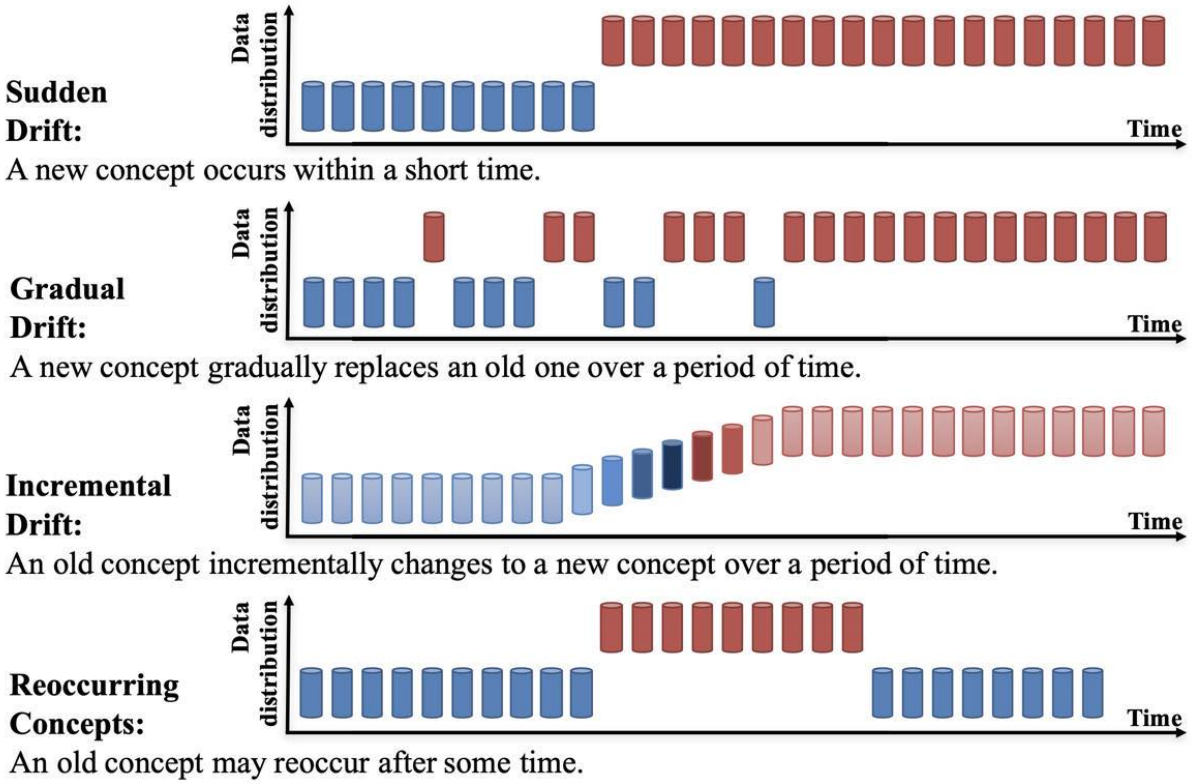


Figure 2.2: Concept Drift Types. Adapted from Lu et al. (2018) [1].

1. **Sudden Drift:** The concept changes suddenly at time $t_0$ [1, 14]:

$$P_t(X,Y) = \begin{cases} P^{(1)}(X,Y), & t < t_0, \\ P^{(2)}(X,Y), & t \geq t_0. \end{cases}$$

   Where $P^{(1)}$ denotes the pre-concept drift joint distribution and

2. **Gradual Drift:** The concept transitions between two distributions over an interval $[t_0, t_1]$ [1, 14]:

$$P_t(X,Y) = (1-\alpha_t)\, P^{(1)}(X,Y) + \alpha_t\, P^{(2)}(X,Y), \quad t \in [t_0, t_1],$$

   where $\alpha_t$ is a monotonically increasing function with $\alpha_{t_0} = 0$ and $\alpha_{t_1} = 1$. In gradual drift, the change is progressive, allowing for a smooth transition between the old and new concepts, which can sometimes be tracked and adapted to more effectively.

3. **Incremental Drift:** The concept evolves continuously over time, modeled as [1, 14]:

$$P_t(X, Y) = P^{(1)}(X, Y) + \delta(t),$$

where $\delta(t)$ represents a small, continuous change with respect to time. Incremental drift reflects ongoing, subtle shifts in the data distribution, requiring models that can adapt incrementally without the need for a complete retraining.

4. **Reoccurring Drift:** A previously observed concept reappears after a period of change [1, 14]:

$$P_{t_r}(X, Y) \approx P^{(1)}(X, Y),$$

for some reoccurrence time $t_r > t_0$. This type of drift occurs when cyclic or seasonal patterns cause old concepts to re-emerge, enabling the reuse of historical models or patterns for improved prediction accuracy.

The examination of concept drift sources and types showed that it is a rigorously defined phenomenon in machine learning [1, 14, 12]. The beginning of this section motivated the need of researching adaptation to concept drift in smart meter data and how concept drift occurs in smart meter data (new home appliance, qualitative differences between training and deployment data, and others). Subsequently, GBDT algorithm on top of which my incrementally learning approaches will be build was presented. Lastly, concept drift was presented in its mathematical formality. The next section presents the key developments of the approaches for tackling concept drift. I finish the section with a deeper explanation of index-based eGBDT the improvement of which is the main contribution of this research.



## 2.2 Incremental Learning Methods

This subsection presents the different clusters of approaches to tackling concept drift in continually received data. Due to the limits of scope, we present only a selection of algorithms deemed either foundational or relevant to our research. The task has been defined as incremental

learning, [1] because the ML model needs to incrementally learn new concepts due to conceptually drifting data. If the data are coming in batches of two to hundreds of instances, the task is called *batch learning*. Alternatively, there can be a single data instance continually received. Such one data-point batch is referred to in the literature as *online learning*. We will begin by presenting the foundational algorithms in the history of incremental learning: namely window-based methods and performance-based methods. Subsequently, we present in greater detail ensemble methods into which eGBDT and AUE2 fall. We are improving on these two methods in the next section.

### 2.2.1 Window-Based Methods

Although many if not most incremental learning approaches employ number of different previously rivaling approaches, the oldest approach we find focuses on *instance selection* which quickly developed into window-based method. The oldest approach named STAGGER uses concept descriptions of instances and instance weighting and forgetting [17]. STAGGER uses Bayes rule to update the weights of specific instances with their concept descriptions where descriptions are initially individual classes of a given feature, but gradually they get more complex.

Although concept descriptions continued to be used for bit longer, the approach of selecting relevant instances as a solution to incremental learning problems pivoted towards *windowing* or *window-based methods*. At its simplest, window of fixed size merely forgets oldest instance when a new instance is processed. Such fixed size windows are explored in [18, 19, 20].

Combination of forgetting and concept descriptions gave rise to FLORA algorithms [21]. There are four versions of the FLORA algorithm all presented in the original paper. They all maintain a window of data over time and calculate statistics of the variables' (or concept description) distributions [21]. The instances which are the most representative of the current window are used for retraining or updating. [2]

More flexible approach than FLORA was published in 2004 [22]. Instead of using only one fixed size window, the Lazarescu et al. version uses three windows simultaneously. The authors

[1] Incremental learning is sometimes called *learning in non-stationary environments* sometimes shortened as *NSE* (such as in the Learn++.NSE algorithm [15]) or alternatively *learning from data streams* [16].

[2] FLORA maintains the set of relevant instances. Whether there will be retraining or updating depends on the ability of the predictive model to update or merely fully retrain.

argue that one window, regardless of size, is not going to fit all types and sources of concept drift. For example, in case of a small incremental change the single fixed size window will react in a stepwise fashion; whereas, multiple window approach can react more smoothly.

Next generation of windowing algorithms made the window sizes flexible. **Ad**ptive **win**dowing (ADWIN) finally drops concept descriptions and focuses only on improving windowing by making its size dependent on concept drift. It creates a window with two sub-widows in continually received data [23]. If the distribution statistics of the two sub-windows differ more than a given threshold, the algorithm triggers full retraining because a concept drift occurred. If the statistics are differing less than a specified threshold, the big window and hence the two sub-windows continue to grow because data distributions have not drifted enough. The ADWIN approach was expanded to improve its performance and runtime into number of variants many of which are compared by Moharram et al. [24].

### 2.2.2 Performance-Based Methods

Another family of approaches solves the problem of incremental learning by looking at the performance of the model. One of the oldest and subsequently most built upon algorithms within this approach is called simply Drift Detection Method (DDM) [25]. It considers the error of an incrementally learning classifier as a Bernoulli random variable, which follows a Binomial distribution. It monitors the probability of misclassifications at time $t$, denoted by $p_t$, and its standard deviation $s_t$ computed as:

$$s_t = \sqrt{\frac{p_t(1 - p_t)}{i}} \tag{2.1}$$

At each time step $t$, the minimum observed values $p_{\min}$ and $s_{\min}$ are updated as follows:

$$\text{if } (p_t + s_t) < (p_{\min} + s_{\min}), \quad \text{then} \quad p_{\min} = p_t, \quad s_{\min} = s_t \tag{2.2}$$

The method defines two thresholds for monitoring the error rate:

- A **warning state** is triggered when:

$$p_t + s_t \geq p_{\min} + 2s_{\min} \tag{2.3}$$

- A **drift state** (indicating detection of significant concept drift) is triggered when:

$$p_t + s_t \geq p_{\min} + 3s_{\min} \tag{2.4}$$

The predicting model is then trained only on instances received during the period between the warning state and the drift state.

DDM developed into many variations. For example, Early Drift Detection Method (EDDM) tracks the distance between two pairs of consecutive misclassifications rather than error rate [26]. EDDM proved to be more efficient in detecting gradual concept drifts [27]. Another variation is Reactive Drift Detection Method (RDDM) which improves the performance of DDM by removing old data instances without loss of accuracy though with a slight delay of drift detection [28]. DDM was also extended to find the specific location of the drift by the local drift degree algorithm (LDD) [29]. Last but not least, DDM was extended by the the Hoeffding bound into Hoeffding DDM (HDDM) [30]. Hoeffding bound, which will be further explored in the next subsection, enables us to give definite statements about data distributions which are infinite; this is highly useful for evolving data streams.

### 2.2.3 Ensemble Methods

Although ensemble methods are sometimes distinguished as either *active* or *passive*, we will focus on passive methods, partly because the two algorithms we improve on are passive ensemble methods and partly because we have explored the active methods. Active ensemble methods are defined as first using one of the drift detection methods, which can be window-based, performance-based or combination of both. Once the drift detection method detects a drift, the ensemble retrains or updates itself. In contrast, passive methods continually update their ensemble with new data rather than actively triggering update or retraining after a drift detection

method triggers such update. In contrast, active ensemble methods have drift adaptation built into the model itself. Sometimes both window-based and performance-based methods are considered active ensemble methods and the whole incremental learning taxonomy collapses into passive and active ensemble methods. [3] While active/passive taxonomy is taken by [31], the taxonomy of widow-based, performance-based and ensemble methods we are using aligns with Bayram et al. They established a thorough systematic literature review of incremental learning approaches [16]. Hence, from now on we will focus on what [31] calls *passive ensemble methods* and Bayram et al. merely *ensemble methods*.

### Hoeffding Tree as Base Learner

A successful family of ensemble methods use Hoeffding trees as base-learners. Hoeffding trees adapt quickly to concept drift by growing sub-tree structures [32]. They use the Hoeffding bound to probabilistically select the best split given a finite data sample, ensuring the split is as good as one made with infinite data given a user specified confidence parameter. [4] Specifically, with probability $1 - \delta$, the true mean differs from the empirical mean by at most:

$$\epsilon = \sqrt{\frac{\ln(1/\delta)}{2n}}$$

where $\epsilon$ is the Hoeffding bound, $n$ the number of observations, and $\delta$ the confidence parameter. Due to their speed, Hoeffding trees are also called Very Fast Decision Trees (VFDT).

The first VFDTs only grew trees, risking infinite size and memory use. The Concept-adapting VFDT (CVFDT) [33] introduces forgetting mechanisms. Subsequent extension of VFDT, Hoeffding Option Trees (HOT), stores multiple promising splits [34].

Although Hoeffding trees are widely and successfully used as base learners, I believe there are at least three reasons why their use for incremental learning is problematic. First, VFDTs are incompatible with models relying on different base learners, such as oblivious decision trees used in CatBoost [35]. Good incremental learning should not require changing successful base learners. Second, VFDTs track the number of seen instances per potential split [32], possibly

[3] That is if the underlying model is an ensemble model. We have not found a case where the underlying method is not an ensemble. It remains unclear how such situation would be called.

[4] In data streams, data is considered infinite.

increasing memory use, especially with larger trees or HOT versions which also track alternative best splits. Moreover, oblivious trees, due to their symmetric structure, are likely more memory-efficient [5]. Thirdly and most importantly, Hoeffding trees require variable memory as they grow and prune sub-structures, which is in conflict with the fixed memory limits of edge computing. Limiting VFDTs to a maximum memory cap constrains their core advantage of growing tree sub-structures. Fortunately, weighting or removing base learners does not depend on Hoeffding trees, and it is this approach that we improve in our research.

### Weighting and Pruning Ensemble Methods

The earliest framework in this approach is the Streaming Ensemble Algorithm (SEA), it laid the groundwork for passive ensemble learning founded on pruning and incremental training of trees directly on data streams [37]. SEA aggregates predictions via simple majority voting of the individual classifiers. This approach builds on off-line ensemble models (non-incrmentally learning models). However, SEA employs it in an incremental batch-learning environment. When a new batch arrives, SEA trains and evaluates a new classifier ($C'$) on the new batch. If there is a space in the ensemble for a new classifier, then $C'$ is automatically added to the ensemble. If the maximum number of classifiers in the ensemble is reached, $C'$ needs to outperform at least one of the current classifiers. Subsequently, the worst performing classifier which is outperformed by $C'$ is pruned and $C'$ is added to the ensemble. SEA thus relies on removing and building new trees. It experimented with weighting, but its results do not show consistent improvement of weighting over mere building and pruning.

A foundational paper which successfully introduced weighting as a strategy for incremental learning is called Accuracy Weighted Ensemble (AWE) [38]. AWE is similar to SEA in maintaining a fixed maximum number of classifiers $K$. Adding and pruning of classifiers is also similar, because the worst classifier is pruned. However, apart from classifier replacement, AWE adjusts weights to each classifier based on how much it outperforms a random classifier.

$$w_i = \text{MSE}_{random\_classifier} - \text{MSE}_{classifier\_i} \tag{2.5}$$

[5] Supported by research into cache-oblivious structures [36]

**Algorithm 1:** Accuracy Weighted Ensemble (AWE)

**Input:** $S$: a dataset of *ChunkSize* from the incoming stream
$K$: the total number of classifiers
$C$: a set of $K$ previously trained classifiers
**Output:** $C$: a set of $K$ classifiers with updated weights

```
Train classifier C' from S;
Compute error rate of C' via cross-validation on S;
Derive weight w' for C' using Equation 2.5;
for each classifier C_i ∈ C do
    Apply C_i on S to derive MSE_i or b_i;
    Compute w_i based on Equation 2.5;
end
C ← the top K weighted classifiers in C ∪ {C'};
return C;
```

Accuracy Updated Ensemble (AUE) presented by Stefanowski et al. builds on AWE by putting Hoeffding trees as its base learners and changing the weighting equation [8]. Stefanowski et al. argue that changing merely the weights of classifiers leads to the problem of "tuning the block size". Which means that the performance of the weighting mechanism is tied to the size of the block (which can be independent from the predicted batch) after which the weights are changed.

Using VFDTs allows updating without such block size tuning. In AUE, the trees will update whenever the Hoeffding bound, given the user inputted confidence parameter, reaches enough data to perform attribute split. Hence, for a series of non-drifting batches both AWE and AUE will not change their weights substantially, but AUE will be able to adapt to it through growing tree sub-structures. In addition, AUE maintains a larger set ("a buffer") of classifiers from which only a subset with the lowest MSE on the latest batch is selected to predict the next batch. Such selecting revives old classifiers in case of recurrent drift. Lastly, Stefanowski et al.l drop the comparison to random classifier during the weight calculation, because "in rapidly changing environments with sudden concept drifts (as the Electricity data set) this threshold can "mute" all ensemble members causing no class to be predicted." The $\epsilon$ in 2.6 is a very small value added in case the classifier is perfect and MSE is 0. The rest of AUE is the same as AWE.

$$w_i = \frac{1}{\text{MSE}_{\_i} + \epsilon} \tag{2.6}$$

Brezinski and Stefanowski, authors of AUE, two years after the publication of AUE published AUE2. AUE2 somewhat improves AUE accuracy while not increasing memory and computational costs [32]. Firstly, they criticize AUE and AWE for using 10-fold cross validation when computing error rate of $C'$ on $S$. AUE2 removes the cross validation and uses 2.6 to immediately calculate the weight of the new classifier using its error; this seems to be the main source of accuracy increase compared to AUE. Secondly, AUE2 drops the buffer set as it improved accuracy ever so slightly while roughly doubling memory usage. Thirdly, AUE2 solves the mentioned problem of potentially infinitely growing Hoeffding trees by dropping the least active leaves to match user inputted memory limits. In addition, the AUE2 paper experimentally shows that updating the weights of only a subset of base learners decreases the overall accuracy while not providing justifiably large memory saving. In the experiment, AUE2 outperforms AUE and AWE on most datasets with having roughly equal computational time and memory requirements. Our second incrementally learning algorithm presented in Section 3.1 is a combination of AUE2 and GBDT adapted for regression tasks.

There are many other weighting approaches which sprouted their own lineages of improved version. For the sake of minimal completeness of related work we will present the main ones. The Dynamic Weighted Majority (DWM) introduced weighting even before AWE (despite AWE not citing DWM) [39]. Outperforming STAGGER, DWM introduced normalization of weights, and a constant multiplicative degradation factor $0 \leq \beta \leq 1$. Thus DWM considers only age in determining classifier's weight. To consider performance as well, DWM was extended into DWM-WIN [40]. Further, Recurring Dynamic Weighted Majority (RDWM) built on DWM by maintaining two ensembles. The primary ensemble performs well on current concepts, while the secondary ensemble consists of the most accurate learners in a long term horizon [41].

Another important algorithm is the Learn++.NSE. Recall that NSE stands for non-stationary environments. Inspired by incrementally learning algorithms for Neural Networks called Learn++ [15], Learn++.NSE weights new data instances before training a new classifier on them. This way the new classifier complements the old classifiers which might understand some instances in the new batch but not all. Sigmoid function which combines classifier's age and accuracy is used to calculate final classifier weights. Learn++.NSE does not drop classifiers

so that it can quickly adapt to recurrent drifts. Since its memory can grow infinitely large, the same authors add forgetting mechanism in their subsequent paper [42]. Learn++.NSE was also developed into two parallel ensembles approach, but unlike RDWM, the Learn++.NSE extension aims at tackling class imbalanced datasets [43]. One of the very few presented algorithms, Learn++.NSE was adapted and tested for regression task under the name On-line Weighted Ensemble (OWE) [44]. We take from Learn++.NSE for our second algorithm in Section3.1 the idea of using sigmoid function to calculate weights. Having presented the foundational papers for solving incremental learning by weighting classifiers, we will now turn to two recent papers which combine classifier training and pruning with GBDT into eGBDT. Our two algorithms are building on top of eGBDT.

Elastic GBDT leverages the already powerful GBDT model but adds a simple incremental learning and pruning. Published first by Wang et al. [6], elastic GBDT prunes by finding the index $m$ of the classifier at which the ensemble as a whole has the lowest Mean Absolute error (MAE):

$$m = \arg\min \mathrm{MAE}(R_m) \tag{2.7}$$

If this index $m$ is lower than some threshold $M$ (line 4 of Algorithm 2), then there is a strong concept drift and the whole ensemble is retrained on the latest batch. If, however, $m \geq M$, then all classifiers with indices greater than $m$ (i.e., $m+1, m+2, \ldots$) are pruned (deleted) (line 8 of Algorithm 2). Then $L$ new trees are fitted on the pseudo-residuals left by $F_m$ (line 9 of Algorithm 2). These $L$ trees are then added to the ensemble at indices $m+1, \ldots, m+L$. Lastly, the updated ensemble makes predictions for the next batch using the relevant data it has at its disposal (usually $D_{\text{batch}}$, but if it has larger memory it can use older data as well).

**Algorithm 2:** Elastic GBDT (eGBDT) for a Single Batch

**Input:** Training dataset $D_{\text{train}}$

New batch $D_{\text{batch}}$

**Output:** Updated GBDT model $F'_M(x)$

Train the initial GBDT model $F(x)$ on $D_{\text{train}}$;
Compute residuals $R_m$ for each tree $m$ on $D_{\text{batch}}$;
Find the best tree index $m$ using Equation 2.7;
**if** $m < M$ **then**
    Retrain the GBDT model $F(x)$ on $D_{\text{batch}}$;
**end**
**else**
    Prune trees after index $m$:
        Update model: $F'_M(x) \leftarrow F_m(x)$;
    Fit $L$ new trees on the pseudo-residuals left by $F_m$ on $D_{\text{batch}}$;
    Add the $L$ new trees into the ensemble:
        Update model: $F'_M(x) \leftarrow F'_M(x) + F_L(x)$;
**end**
Predict the next batch using $F'_M(x)$ based on $D_{\text{batch}}$ data;
**return** $F'_M(x)$;

In Wang et al.'s experiment, index-based eGBDT outperformed all competing algorithms including the original Learn++.NSE, two newer versions of HDDM, and one newer version of AUE (we did not present the newer versions, only the foundational ones) [6]. Furthermore, Chen et al. used index-based eGBDT to detect bot attacks on internet of things with similar success [7]. However, index-based eGBDT was outperformed by OnlineAUE (newer version of AUE) on one type of data in Chen et al.'s experiment. Because of its persuasive performance, we have decided to adapt eGBDT for the regression task of energy time series data prediction. However, the adaptation needs to solve two issues while ideally increasing model's accuracy. Firstly, index-based eGBDT needs to be adapted for regression rather than classification. Secondly, its memory has to be clipped. Since in a string of non-drifting batches, index-based eGBDT continually builds new weak learners, but does not prune any old ones, its memory usage can

infinitely grow. Our solutions as well as experimental testing are presented in the next section.

# Chapter 3

# Results

We have adapted index-based eGBDT into two versions for regression tasks. Both versions use regression trees as its base learners and prune based only on tree's contribution to the reduction of error disregarding tree's index. The first version is called contribution-based eGBDT (cbeGBDT). The second version not only prunes but also weights its trees on a contribution-based mechanism with weighting mechanism similar to AUE2; hence we call it Accuracy Updated eGBDT (AUeGBDT). We first present the two algorithms including their order of complexity. We then test their accuracy in an experiment comparing them to: baseline GBDT model which does not retrain, baseline model which fully retrains on the latest batch, index-based eGBDT just presented and index-based eGBDT with constant number of trees.

## 3.1 Theoretical Results

Contribution-based eGBDT tracks the error improvement $I_i$ for each tree $i$ with multiplier $\gamma$ (found by line search A.3) as it is added to the cumulative prediction (lines 3-6 in Algorithm 3). The worst $K$ trees are considered non-contributing and are pruned from the ensemble (lines 9-10). The calculations determining K are presented in the next paragraph.[1] The new smaller ensemble is then tested on the current batch and its residuals are obtained (line 15). $K$ new trees are then trained on these residuals (line 16) and added to the ensemble (line 17). If $M = K$ i.e., all trees were deleted the prediction is the mean of $y$ which is one of the initial prediction of standard GBRT model [45]. Finally the updated ensemble makes predictions for the next batch (line 19).

As was mentioned, $K$ can be determined in a number of ways (line 8). We have experimented with exponential moving average (EMA), but the early experiments already showed that this

[1] We experimented with a threshold of contribution; e.g. 0 which would mean that all tress which increase the error pruned but also other thresholds. This led to lower performance than pruning based on ranking. Arguably, threshold value is less dependent on data than ranking which better determines outdated trees relative to the rest of the ensemble.

method leads to high per-batch pruning and thus high computational cost while having low accuracy. The second method we tried is a ratio between the whole ensemble's error on the current and the last batch multiplied by a tuned multiplier. The result is rounded to an integer (ratio method). The third method is a four ratios long chain of such batch-to-batch error ratios. Each with its own tuned multiplier. The results from each ratio are added together and then a tuned multiplier is applied. The final result is also rounded to an integer value (contribution ratio). While ratio method has only one hyperparameter - the multiplier-, contribution ratio method has 4 - 3 for each ratio and the last multiplier for the sum.

**Algorithm 3:** Contribution-Based eGBDT for a Single Batch

**Input:** Trained GBDT model $F_M(x)$ with $M$ trees notated as $h_i$

New batch $D_{\text{batch}} = \{X, y\}$

Tuned hyperparameter: Learning rate $\gamma$ , Threshold of retraining $T$, choice of calculation method for $K$ with its own parameters

**Output:** Updated GBDT model $F'_M(x)$

Initialize prediction with mean: $\hat{y} \leftarrow \bar{y}$;

Compute initial error: $E_0 \leftarrow \text{MAE}(y, \hat{y})$;

**for** *each tree* $T_i$ *in the ensemble* **do**

    Predict: $\hat{y}_i \leftarrow \hat{y}_{i-1} + \gamma \cdot h_i(X)$;

    Compute error: $E_i \leftarrow \text{RMSE}(y, \hat{y}_i)$;

    Compute improvement: $I_i \leftarrow E_{i-1} - E_i$;

**end**

Determine $K$ using a function of choice;

Prune least contributing $K$ trees:

    Update model: $F'_M(x) \leftarrow F_M(x) \setminus \{h_i \in K\}$;

Test $F'_{M-K}(x)$ and compute residuals $R$;

Train $K$ new trees on residuals $R$;

Update model: $F'_M(x) \leftarrow F'_{M-K}(x) \cup \{\text{new } K \text{ trees}\}$;

Predict the next batch using $F'_M(x)$ based on $D_{\text{batch}}$ data;

**return** $F'_M(x)$;

The order of complexity of contribution-based eGBDT can be analyzed in terms of the

number of trees $M$, the number of samples in a batch $N$, and the depth $d$ of each decision tree. For each batch, the algorithm evaluates all $M$ trees with their depth $d$ in the ensemble to compute their contribution, resulting in a computational complexity of $O(M \cdot N \cdot d)$ due to tree traversal per sample (line 3). At line 4, it computes error for each tree for each sample, but since predictions of the samples are now in vectorized form this constant time for each tree $O(M$. Equally at line 5, the contribution of each tree is calculated by subtraction of two numbers; hence it is at constant time for each tree $O(M$. Together lines 3-6 cost:

$$O\big(MNd + 2M\big).$$

The leading term is the ensemble traversal $O(MNd)$. It scales with both the batch size $N$ and the ensemble size $M$, while the additive term $O(2M)$ is dominated.

The pruning step (lines 9-10) involves ranking contributions, which takes $O(M \log M)$ if using merge sort and the calculation of $K$ [46]. Ratio method involves two multiplications which is computationally complexity of a constant $O(2)$. Weighted ratio method involves three multiplications of ratios, one addition of the total product and one final multiplication. This also leads to a constant complexity of $O(5)$. Constants are negligible compared to the other terms.

If there is pruning of one tree or more i.e., $K > 0$, the next step inevitably costs $O(M \cdot N \cdot d)$. This is in case of the full retraining of all $M$ trees, but also in case of mere training of $K$ trees, because firstly the pseudo-residuals on which the $K$ trees will be trained need to be calculated by the prediction of $M - K$ trees. With pseud-residuals established by $O((M - K) \cdot N \cdot d)$, $K$ trees will be trained $O(K \cdot N \cdot d)$. Hence, if there is pruning of even a single tree the computational complexity is the same as if the model were fully retraining. This means that in case of pruning the contribution-based eGBDT requires computational costs for both ranking and full model retraining. This leads to extra costs of $O(c + M \log M)$ compared to a model which retrains fully on every batch. Together with contribution calculation and in case of pruning contribution e-GBDT costs:

$$O\big(MNd + 2M\big) + O(M \log M) + O(MNd) \;=\; O\big(2MNd + 2M + M \log M\big).$$

Hence, $O(M \cdot N \cdot d)$ remains the leading term just as it is in GBDT.

Although, during development, cbeGBDT outperformed index-based GBDT, its adaptability was not as granular as it was desirable. For ensembles of 20, 30 or 40 trees removing whole trees even one tree can have a big impact on the ensemble's prediction. Moreover, the model disregards indexes when pruning trees. This leads to the the possibility of removing the first tree which; however, can have very large impact on the prediction. Vis versa, not removing a tree could lead to the model not adapting to smaller incremental drifts. The strategy of removing and building new trees leads to desired adaptive granularity for models with hundreds of trees. For example, Wang et al. used 250 initial trees with 25 incrementally trained trees while Chen et al used even 10.000 trees [6]. However, edge computing limitations demand that the ensemble is capped at dozens rather than hundreds of trees. Hence, to achieve greater adaptive granularity I decided to introduce weighting to the contribution-based eGBDT. Inspired by the successful AUE (see Section2.2.3) and Learn.NSE++ (Section2.2.3), below we present the Accuracy Updated Elastic GBDT.

**Algorithm 4:** Relative Contribution–Based Weight Update for a Single Batch

**Input:** Trained GBDT model $F_M(x)$ with $M$ trees notated as $h_i$

Set of per–tree weights $\{w_i\}_{i=1}^{M}$ all initialized at 1

New batch $D_{\text{batch}} = \{X, y\}$ of $N$ examples,

Tuned hyperparameter: learning rate $\gamma$, sigmoid slope $k$, weight–scale $w_{\text{scale}}$,

weight update strength $\rho$, pruning threshold $\theta$, small constant $\varepsilon$

**Output:** Updated weights $\{w_i''\}_{i \notin \mathcal{R}}$ and index set $\mathcal{R}$ of removed trees

Initialize prediction with mean: $\hat{y} \leftarrow \bar{y}$;

Compute initial error: $E_0 \leftarrow \text{RMSE}(y, \hat{y})$;

**for** *each tree $T_i$ in the ensemble* **do**

  Predict: $\hat{y}_i \leftarrow \hat{y}_{i-1} + \gamma\, w_i\, T_i(X)$;

  Compute error: $E_i \leftarrow \text{RMSE}(y, \hat{y}_i)$;

  Compute relative improvement (contribution):;

$$\text{cont}_i = \frac{E_{i-1} - E_i}{E_{i-1} + \varepsilon}$$

  ;

**end**

Convert the raw contributions directly into proposed weights using a sigmoid transformation:;

$$w_i' = \frac{w_{\text{scale}}}{1 + \exp\left(-k\, \text{cont}_i\right)}$$

Identify and remove trees whose proposed weight falls below the fixed threshold:;

$$\mathcal{R} = \{\, i \mid w_i' < \theta \}$$

Remove these trees and their weights from the ensemble, and let $K = |\mathcal{R}|$.;

For every remaining tree $i \notin \mathcal{R}$ blend the previous weight toward the proposal:;

$$w_i'' \;=\; (1 - \rho)\, w_i \;+\; \rho\, w_i'$$

If $K > 0$, update the current prediction on $X$ using the blended weights for the retained ensemble:;

- Sigmoid slope $k$: controls the slope of the sigmoid $\frac{w_{\text{scale}}}{1+\exp(-k\,\text{cont}_i)}$. High $k$ yields a near-step function (sharp distinction), a low $k$ results in a more gradual curve.

- Weight-scale $w_{\text{scale}}$: sets the upper bound of the updated weight $w_i''$. If $w_{\text{scale}} = 1$, weights lie in $(0, 1)$; setting it above 1 allows strong contributors to be weighted more than 1. Given the default weight of 1 for every tree, setting $w_{\text{scale}} = 2$ leads to contributing trees having weights between 1 and 2, but trees which worsen the prediction will have their weights reduced below 1.

- Weight update strength $\rho$: determines how quickly old weights adapt toward the newly estimated weights. A small $\rho$ keeps the ensemble memory strong (slow adaptation); a large $\rho$ moves weights rapidly toward the latest estimates.

When a new batch of size $N$ arrives, the total computation breaks down as follows:

*Evaluate existing trees and their contributions.* This is the same as in the contribution-based eGBDT. Each of the $N$ examples must traverse all $M$ trees and there are two calculations for each tree. Hence:

$$O\big(MNd + 2M\big).$$

*Weight update (Alg. 4, lines 4–6)* involves three constant-time operations for each of the $M$ trees:

- computing the sigmoid $w_i'$,

- checking $w_i' < \theta$ to mark pruning candidates,

- blending $w_i'' = (1-\rho)w_i + \rho w_i'$.

Since each is $O(1)$ per tree, updating all weights costs

$$O(3M).$$

The remaining costs are the same as in the contribution-based eGBDT (Alg. 3):

$$\underbrace{O\big(M \log M\big)}_{\text{ranking (lines 9–10)}} \;+\; \underbrace{O\big((M-K)\,N\,d + K\,N\,d\big)}_{\text{retraining (lines 15–17)}}.$$

Putting everything together, the computational per-batch complexity of Alg. 4 is the same as in the contribution-based eGBDT (Alg. 3) but with the addition of $O(3M)$:

$$\underbrace{O\big(M\,N\,d + 2M\big)}_{\substack{\text{contribution \& pruning}\\ \text{(Alg. 4)}}} + \underbrace{O(3M)}_{\substack{\text{weight update}\\ \text{(Alg. 4, lines 4–6)}}} + \underbrace{O\big(M \log M\big)}_{\substack{\text{ranking}\\ \text{(Alg. 3, lines 9–10)}}} + \underbrace{O\big(M\,N\,d\big)}_{\substack{\text{retraining}\\ \text{(Alg. 3, lines 15–17)}}} .$$

Again, the ensemble traversal term $O(MNd)$ is the leading term, because it scales with both the batch size $N$ and the ensemble size $M$. The remaining terms grow only linearly or logarithmically with $M$, so the incremental extension of weight update and new tree training does not create more computationally complex leading term.

There are three significant differences between cbeGBDT and AUeGBDT. Firstly, the contribution is calculated as a relative improvement on the inherited pseudo-residuals (line 6 Alg. 4) rather than absolute improvement on them (line 6 Alg. 3). This is because empirically the results were shown to be slightly better for each algorithm to calculate it thus. My hypotyhesis is that cbeGBDT maintains stronger interdependence between the trees; whereas AUeGBDT breaks this interdependence of trees even further with the introduction of weights. Dividing the improvement by the original pseudo-residuals reflects the relatively smaller tree interdependence.

Secondly, there is the weight update which introduced three new hyperparameters and the sigmoid activation function. As was explained earlier, this is aimed to allow incremental learning at greater granularity then pruning and building trees allows. It further expands the space for tuning because it introduces three new hyperparamaters to tune. One can also change the activation function which creates a new hyperparameter. Expanding tuning space is not strictly desirable, because ideally the tuning would take place within the edge device, but with larger optimization space optimal tuning becomes unfeasible.

Finally, the AUeGBDT is much less likely to prune and build new trees, because the weight calculation already introduces high degree of adaptability. Therefore, the additional retraining computational cost $O(MNd)$ will happen sparsely. However the retraining term has the same complexity as the batch traversal term.

## 3.2 Experiment

In this section the introduced algorithms are applied on the smart meter time series dataset. It starts with the description of the dataset and preprocessing steps. Subsequently, the compared models with, the error metrics, training and hyperparameter tuning are presented. Finally, the results are plotted.

### 3.2.1 Dataset and Preprocessing

The smart meter time series dataset from Kaggle [47] was chosen due to its quality and public availability. Since the primary contribution of this research is theoretical, the empirical part follows the core principles of time series data preprocessing rather than introducing novel approaches for improvement.

The dataset covers energy consumption from December 2006 to November 2010 with momentary readings stored every minute. It consists of 9 columns. Two describe the time of the recording: one is a date and another is the time of day. These were collapsed these into a single datetime column. The only other original variable we kept is `global_active_power`. The six remaining columns were dropped, namely: `sub_metering_1`, 2 and 3 which corresponded to the different phases, `global_reactive_power`, voltage and `global_intensity`. The reason for dropping all other columns is that we wanted to keep feature engineering as simple as possible. While this might have lower predictive accuracy of all models, the main task is not to achieve the highest accuracy on this particular dataset, but rather to compare different models and their adaptation to concept drift as it occurs in energy data. For that all that is needed is a representative dataset and not necessarily highly complicated one.

Arguably, `global_active_power` measured in kW is not the most suitable physical quantity to conduct experiments on, because its values are momentary and as such its values can be haphazard. Ideally, the predicted quantity would be in kWh, but such high quality energy data are rarely public and in the interest of replicability we chose to predict public kW dataset rather than undisclosed kWh dataset. Nonetheless, the relatively high 1-minute granularity partially makes up for the disadvantage of the momentary value. In EU, smart meter readings are recorded in a 15-minute interval while in the UK the interval is 30 minutes [48, 49, 50]. To compensate for

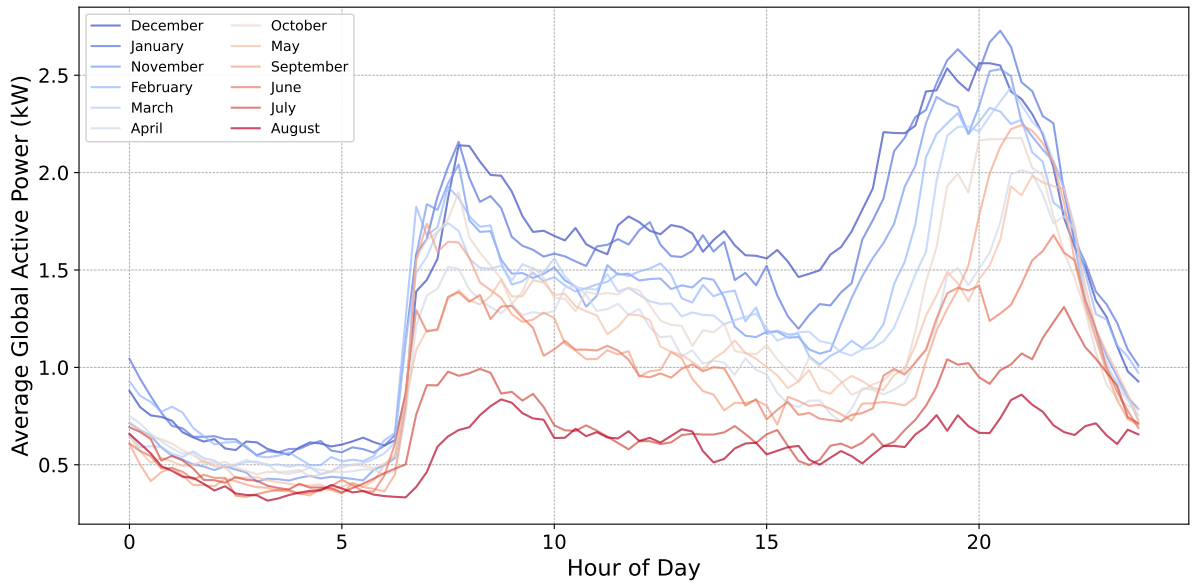


Figure 3.1: Daily Energy Consumption Profile for Months

the haphazardness of momentary values and to imitate EU energy distributing company which could be interested in 15-minute predictions the data were aggregated into 15-minute readings.

However, before the aggregation, the issue of missing values neeeded to be dealt with. The dataset had total of 2075259 1-minute measurements with 25979 missing values. This corresponds to roughly 1.25% of values. We simply interpolated the missing values as a mean of the last recorded measurement and the next recorder measurement. Only afterwards was the 15-minute aggregation applied. Thus processed data need to be explored to establish the presence of concept drift.

In Figure 3.1 the daily profiles for every calendar month are averaged. If data from that month were collected repeatedly - e.g. 4 times in the 4-year dataset -, these are also averaged. The daily profiles of the processed dataset are analogous to daily profiles visualized elsewhere [51, 52]. They have the typical morning and and evening peaks when people use their house appliances the most and mid-day trough when people are at work. Furthermore, the Figure 3.2 shows the whole dataset after 15-minute aggregation was applied. We can clearly see the seasonality of energy consumption. If the model does not train on year or more of data, then it will be difficult for it to predict the conceptually drifted data as previously unseen season comes.

Next step is feature engineering. Fourier Transform, PACF and ACF are applied to see if

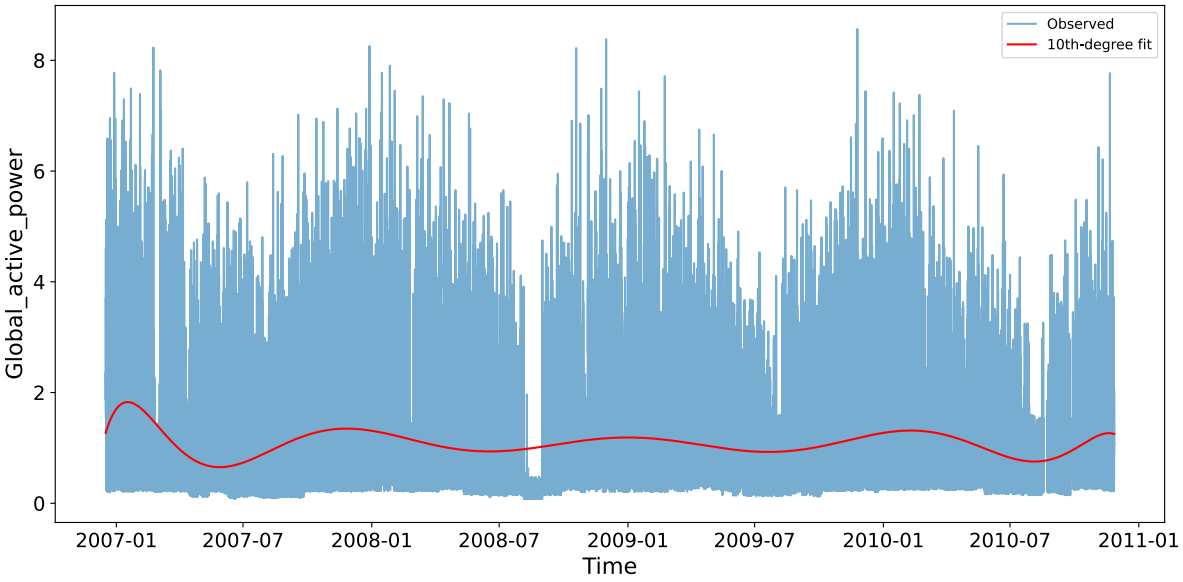


Figure 3.2: Global Active Power with 15-Aggregation and Polynomial Line of Best Fit

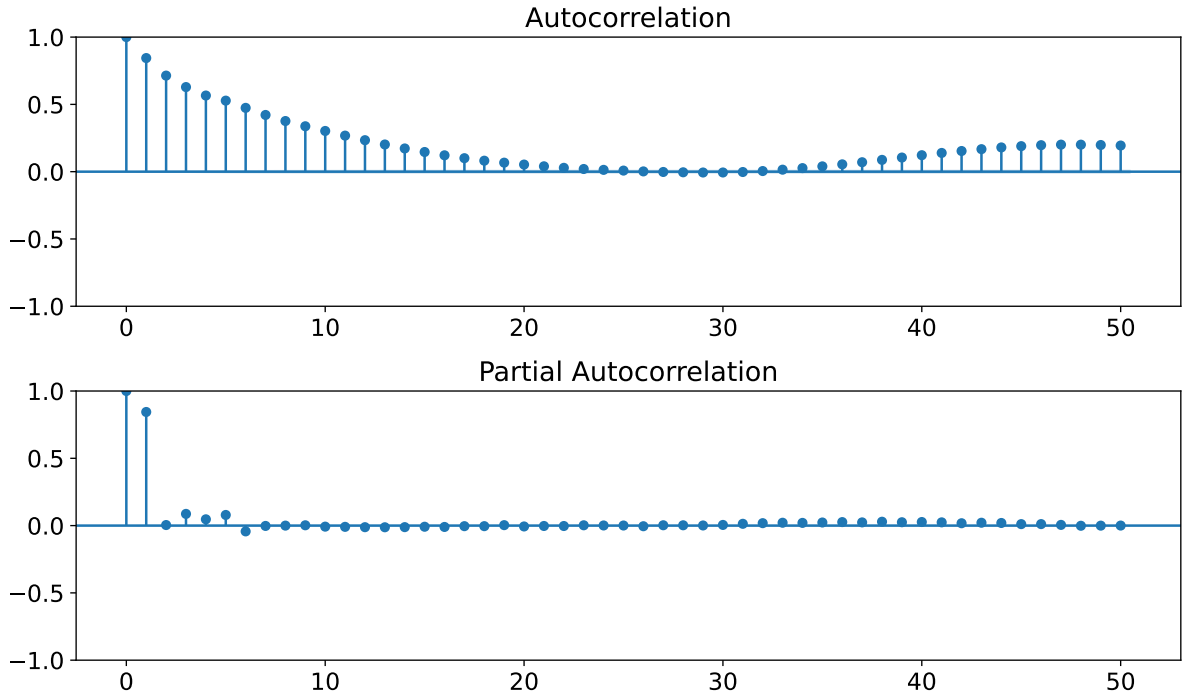


Figure 3.3: PACF and ACF

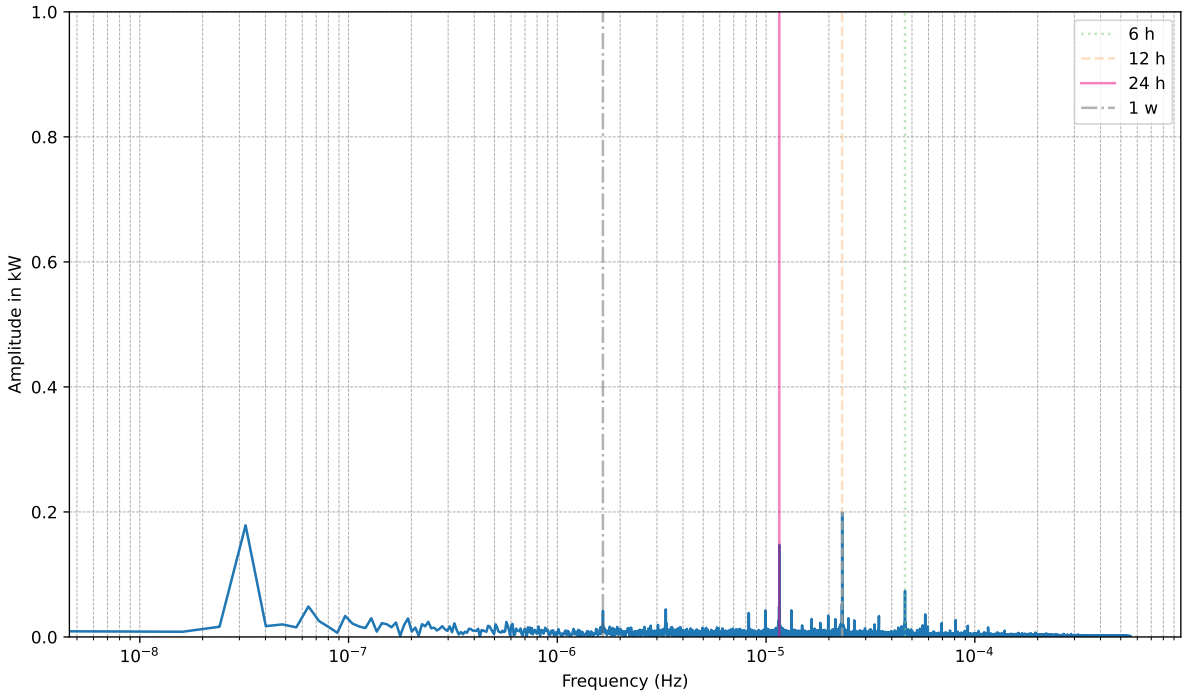


Figure 3.4: Fourier Transform

the data are not stochastic and which lags should be engineered [53, 54]. The ACF (Figure 3.3) shows some continuity of the data up until roughly 15th data-point in past which corresponds to 3 hours and 45 minutes. This supports the claim that there is a continuity of past data influencing the current data and the whole time series is not stochastic.

The PACF plot however tells little of substance and perhaps even suggests there is stochasticity in the data. The fact that only lag of 1 data point is significant is problematic. There is likely no energy-consuming activity which repeats with 15-minute interval. Rather many energy-consuming activities last 15 minutes. It is these roughly 15-minute activities which are picked up by the lag of 1 in the PACF plot. We have decided not to include the lag of 1. While it might be highly significant it would likely lead the model to simply predict the next value to be the same as the preceding value. Why exactly this might happen will be explained after establishing the lag selection using Fourier Transform.

Taking and plotting Fourier Transform of the dataset decomposes the original time series into the periods of the different signals which are in a superposition in the original data; assuming the original time series is a superposition of a number of different signals. Lag feature engineering

with Fourier Transform thus means choosing whichever period is identified as a strong signal in the plot [55]. Indeed, Fourier Transform of the chosen dataset provides better information for determining lag feature engineering than PACF and ACF (see Figure 3.4). As we can see the signal slightly more frequent than $10^{-5}$ is strong. This corresponds to one day period (pink full line).[2] 12 hour signal is even a bit stronger (yellow dashed line). Another strong signal is 6 hours (light green dotted line). We also decided to use 1 week lag even though its signal is relatively weak (grey dashdot line). Nevertheless, some energy consuming activities repeat with weekly basis or are associated with a specific day of the week. The most prominent signal which occurs a bit more frequently than $10^{-8}$ is not included into the lag selection, because it corresponds a to a year. Such a lag is impossible to store in a limited memory edge device. In contrast, month (1 through 12) is used as a static feature (as opposed to lag). Hence, when a new row of data arrives the algorithm knows the month. Unlike lag, which requires 12-month large memory storage, knowing the month of the row of data is an immediate value which does not create any such memory requirements.

Since the forecast horizon is the whole day and 6 and 12 hour lags are used, the recursive time series forecasting method is used [56]. Recursive time series forecasting uses past predictions as features for subsequent predictions. One day of 15-minute predictions corresponds to 96 data-points. Hence, with 6-hour lag the prediction of first 23 data-points use recorded 6-hour lag data. However, for the remaining 73 data-points the numbers for 6-hour lag feature are the already made predictions. Because the 6-hour lag feature 15 hours from now is the outcome variable 9 hours from now and so it not yet recorded.

For this reason the lag of 1 data point mentioned two paragraphs ago was not included. Highly important feature - the lag of 1 - which however is for 95 out of 96 data-points a result of previous predictions propagates error and thus increases it. Even though for the first prediction lag of 1 will likely reduce error, because in 15 minutes the next value is highly correlated with the immediately preceding value as was shown by the PACF (Figure 3.3).

In addition, the recursive time series forecasting creates a new question regarding the features for trees' contribution calculation. Currently, individual tree contribution calculation for both

[2]The period in seconds is attained using $1/f$. For example, $1/10^{-5}$ results in 100.000 seconds which is a bit over 27 hours.

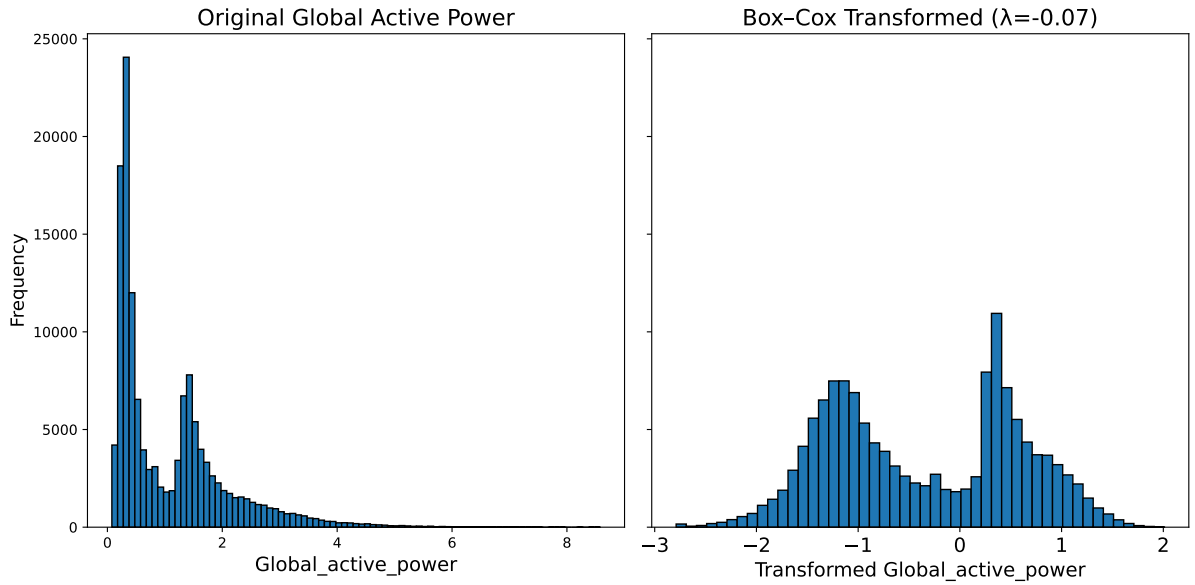


Figure 3.5: Histograms of the Outcome Variable Before and After Applying Box-Cox Transformation

Alg. 3 and Alg. 4 is calculated based on recorded rather than recursively predicted features. Since the contributions are calculated only once the algorithm receives the recorded new data batch, all features for all 96 data-points are available. Hence, there is no need for recursive forecasting. Instead, contribution of each tree is calculated based on every tree's reduction of error given the recorded features rather than recursively predicted features. While this approach establishes the contribution of each tree on real data, it differs from the prediction for next batch where recursive forecasting is used again. However, using recursive forecasting in contribution calculation calculates trees' contribution given predicted and therefore incorrect features. The question of whether to use recursive forecasting, recorded data or some combination of the two in contribution calculation is reopened in Discussion Section 4.

Another preprocessing step was the Box–Cox transformation. The original consumption data are highly left-skewed: the histogram in Figure 3.5 shows two clear peaks—one between about 0.3 kW and 0.5 kW and another around 1.5 kW—with a long tail extending up to 8 kW. Training directly on such skewed data risks over-fitting the small-value region and under-fitting the longer tail. By applying a Box–Cox power transform [57], the distribution becomes more symmetric: the two original peaks map to approximately –0.8 and 0.4 in the transformed scale (see Figure 3.5), stabilizing variance and reducing skewness. The transformation should enable

models to fit well to both modes of the data [58]. The final predictions of the models will be inverted to be on the scale of the original values. The scipy implementation of the Box–Cox transformation was used [59]. The transformation is defined as:

$$y^{(\lambda)} = \begin{cases} \dfrac{y^{\lambda} - 1}{\lambda}, & \lambda \neq 0, \\ \ln(y), & \lambda = 0. \end{cases}$$

For our data, the estimated parameter was:

$$\lambda = -0.0690.$$

$\lambda$ was determined by maximizing the log-likelihood of the transformed data under the assumption of normality using the scipy implementation of the function [59].

The last preprocessing step was changing the time series data into a dataset which can be used for machine learning model training and prediction. This is a simple data manipulation task. Assuming, there is lag of 1 of the outcome variable $y$ at time $t$ it is turned into a feature $X$ at time $t+1$ for predicting $y_{t+1}$. Such data manipulation is called the *sliding window method* and it is commonly used in time series data preprocessing [60, 61]. Of course, this done according to the chosen lags.

Thanks to the Fourier Transform, it was argued that the data are structured around periodical energy use rather than being highly stochastic and therefore unpredictable. The Fourier Transform also led us to engineer four lagged versions of the target variable at 6, 12, and 24 hours, as well as a week-long lag. In addition, the following static time features were included: hour, minute, day, and month (where *day* refers to the day of the month). Although the current selection of time lags may be naive and there is a flourishing research exploring more nuanced procedures for choosing the right time lags [62], the main research goal of this thesis is the improvement of incremental learning algorithms rather than attaining the highest possible accuracy on this specific dataset. Because of that the conducted preprocessing steps including the recursive forecasting are following established foundational time series procedures. Similarly, we are not testing for stationarity of the time series. Good incremental learning algorithm

should have strong predictive accuracy regardless of the stationarity of data. That is indeed why Learn++.NSE has the "non-stationary environment" in its name. On the other hand, if the data were completely or largely stochastic, then comparing models on it would not yield representative results. However, it has been argued by e.g. taking the Fourier Transform that the dataset is not too stochastic and that such model comparison is feasible.

### 3.2.2 Models and Error Metrics

Seven models are compared on four different error metrics (MAE, RMSE, MSE, MAPE). The model trainings and hyperparameter tunings are highly limited to simulate the limitations of edge computing. We now briefly present each of the models, its training and hyperparameter tuning.

Out of the seven models one is a called *Plain*, because it is a GBDT model which is left unchanged once it has trained on the training data. Then there are five models which received initial training but they also learn incrementally. Finally, there is the *Retrain* model which also received initial training but it retrains fully on every batch while in use.

One of the incremental learning algorithms is the index-based elastic GBDT (Alg. 2). It is abbreviated in the results figures as *Index-Stat*, because it trains a static number of incremental trees on every batch, if it does not retrain itself fully on it. In contrast, *Index-Dyn* is an abbreviation for a model which works the same way as index-based elastic GBDT with two key differences. It has a retrain threshold equal to 0 and it does not train new trees on every batch. It too finds the index of the tree with the least error on the current batch and it too prunes all remaining trees. But here Index-Stat would be retraining if the index is too low. Whereas, Index-Dyn regardless of the index trains as many trees as were deleted on the current batch. If the index is the initial guess than the whole ensemble is retrained; otherwise, some trees remain untouched. Index-Dyn is thus our idea of memory stable elastic GBDT as introduced by Wang et al [6].

Another two incremental learning algorithms are the contribution-based elastic GBDTs. The one abbreviated as *Contrib* in the results figures uses weighted contribution ratio to determine how many trees to prune ($K$ in the Alg. 3 explanation). The second is abbreviated as *Ratio* in

the figures and uses the two batch ratio to determine $K$.

The last incremental learning algorithm is abbreviated in the result figures as *Accuracy*. It is the Alg. 4.

The incremental learning algorithms have two sets of hyperparameters to tune. The first set defines the GBDT model on which the incremental learning extension is built. The second set is associated with incremental learning mechanism. Thus, at first trained and using grid search tuned the underlying GBDT model on three weeks of data. Subsequently, thus tuned and trained GBDT model was used as a basis for each incremental learning model. All incremental models, including the Plain model, thus used six weeks of learning data. The difference is that the Plain model used in the model comparison received all six weeks of training data in one training; whereas, incrmental learning models trained and tuned in two phases each consisting of three weeks.

The choice of six weeks on 4-year long time series is intended to simulate the limited edge computing memory which might be even stricter. However, if only 3 or 4 weeks of data were used for the training and two sets of hyperparameter tuning, the tuning would hardly be meaningful. Arguably, even now the models are not trained and tuned enough. In the tradeoff between proper training and tuning versus credible simulation of edge computing environment, this research opted for a compromise leaning towards credible simulation of edge computing limitations.

The base GBDT model was trained and tuned using scikit-learn's `TimeSeriesSplit` [63]. We used two splits, each holding one week of data as the test set with the following parameter grid:

```
param_grid = {
    'n_estimators':     [30, 50, 70],
    'max_depth':        [3, 5],
    'learning_rate':    [0.05, 0.1]
}
```

Results of the tuned base GBDT model:

- 2 folds for each of 12 candidates, totalling 24 fits

- Best params: {learning_rate: 0.1, max_depth: 3, n_estimators: 50}
- CV-RMSE: 0.8704

Thus tuned GBDT model was then used as the base for each incremental learning variant. The following hyperparameter grids were defined for the different incremental learning models:

```
# 1) Contribution-based elastic GBDT (Contrib)
contrib_grid = {
    "ratio_fixed_multiplier":[1, 1.25, 1.5],
    "alpha_multiplier":      [0.4, 0.7],
    "beta_multiplier":       [0.1, 0.3],
    "gamma_multiplier":      [0.1],
}
ratio_grid = {
    "ratio_fixed_multiplier":[1.2, 1.5, 1.8],
}
accuracy_grid = {
    "sigmoid_k":       [ 1.6, 2, 2.4],
    "fixed_threshold":[0.2, 0.6],
    "ratio_power":     [0.2, 0.6, 1],
    "w_scale":         [2]
}
static_grid = {
    "fixed_rebuild_count":[2, 5, 8],
    "m_threshold":          [50],
    "error_power":          [2]
}
```

The tuning settings for incremental learning models were the same as the training settings for the base GBDT model. However, the following three weeks were taken (2016 data points). There were also with two splits applied using TimeSeriesSplit and one week test size. These are

the resulting best models:

**Best Tuned Incremental Models**

- **AccuracyUpdatedGBDT**: {`sigmoid_k`: 1.6, `fixed_threshold`: 0.2, `ratio_power`: 0.2, `w_scale`: 2}, CV-RMSE: 0.7845
- **Ratio-Based**:`ratio_fixed_multiplier`: 1.5}, CV-RMSE: 0.7829
- **Index-Static**: {`fixed_rebuild_count`: 8}, CV-RMSE: 0.9321
- **Contrib-Weighted**: {`m_threshold`: 0, `ratio_fixed_multiplier`: 1, `alpha_multiplier`: 0.4, `beta_multiplier`: 0.1, `gamma_multiplier`: 0.1}, CV-RMSE: 0.7839

Finally, a new plain GBDT model (abbreviated in the figures as *Plain*) was trained on six weeks. Its hyperparamater grid was a bit expanded so as to be more comparable to the incremental learning models which had two sets of hyperparamaters and therefore larger optimization space. Six weeks are together 4 032 data points. `TimeSeriesSplit` was used with four splits. This is supposed to be analogous to the incremental learning models which used two splits for base GBDT and two splits for incremental learning extension parameter tuning

```
param_grid = {
    'n_estimators': [20, 30, 50, 70],
    'max_depth':    [3, 4, 5],
    'learning_rate':[0.01, 0.05, 0.1]
}
```

- Fitting 4 folds for each of 50 candidates, totaling 200 fits
- Best params: {`learning_rate`: 0.05, `max_depth`: 5, `n_estimators`: 30}
- CV-RMSE: 0.8189

The reason for choosing RMSE as an error metric to minimize is the compromise between considering large and small errors. Large errors in energy consumption forecasting can lead to spikes of shortage or oversupply of energy which can become bigger problems then aggregated

small deviations of the same absolute value. On the other hand, MSE penalizes large errors perhaps too severely. All in all, the choice of error metric depends on the task and here is no other specific task than model comparison. That is why four metrics are included namely: RMSE, MAE, MAPE, and MSE; though the choice of minimizing RMSE is somewhat arbitrary. Good incremental learning method should be oblivious to the particular error metric and much energy forecasting research publishes numerous metrics [64, 65].

After all models were trained and their parameters have been tuned as described, they were used to predict the remainder of the data in 96 data-point batches which correspond to one day. This a meaningful forecast horizon, because a key energy market is the day-ahead market (sometimes called *spot market*) [66]. Again, this is a compromise between very short forecast horizon of hours or even minutes, both of which are highly useful for intra-day energy trading and flexibility services supply, and longer weeks or months forecast horizons which determine equally long energy contracts [67].

The longer forecast horizon during parameter tuning (one week) and a shorter forecast horizon (one day) during batch learning were picked to avoid overfitting during parameter tuning. On the other hand, such short horizon during incremental learning can lead to undesired overfitting for incremental learning models, if the concept drift is not strong. In such situation having stronger bias and thus not forgetting recurrent concepts is useful. The short forecast horizon thus advantages Plain model as opposed to Retrain model or to lesser extent incrementally learning models.

To better compare the models, two other testing setups are included. In the second setup, all models train six weeks as in the first setup, but the forecast horizon is extended from 1 day to 28 days. This should advantage Retrain and to lesser extent incremental learning models whose newly trained trees should benefit from a larger dataset to train on. In the third setup, the base model receives 2 years of training. Incremental learning models will receive no extra training and their parameters will be the same as those find by tuning in the first setup. The forecast horizon for this third setup will be a week. This setup should benefit the less flexible model such as the Plain model, because there will be enough data to train on including seasonal trends. In contrast, the retrain model will forget all that has learned by retraining already on first batch.

### 3.2.3 Experimental Results

In this section the results ftom the three experimental setups are presented. The results of the first setup - which imitates the edge computing and energy distribution company most faithfully - are presented have bar plots for all error metrics. The remaining two setups and average statistics across the three setups are presented only by tables. Furthermore, MAE and RMSE evolution over the batches is presented for the second setup (6 weeks training, 4 weeks forecast horizon). The remaining two setups have too many data-points (batches) to display the error evolution in a readable format. The main conclusion of this section is that there is no one dominating model.

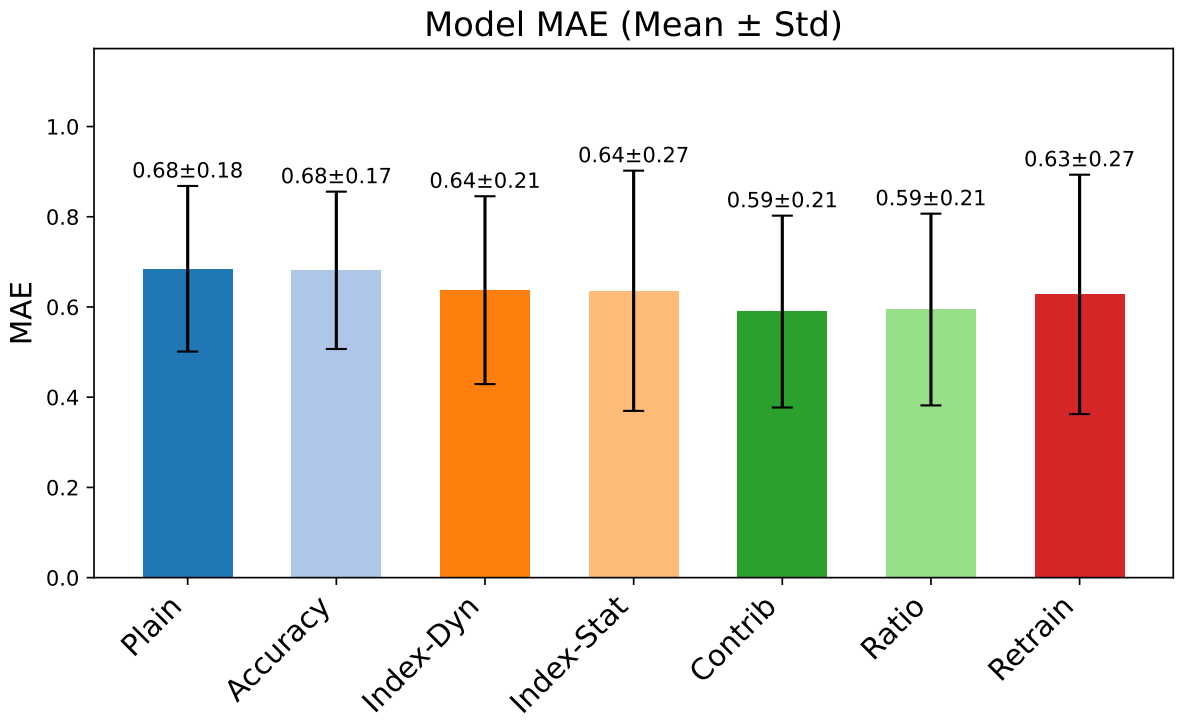


Figure 3.6: Model MAE with Six Weeks of Training Data

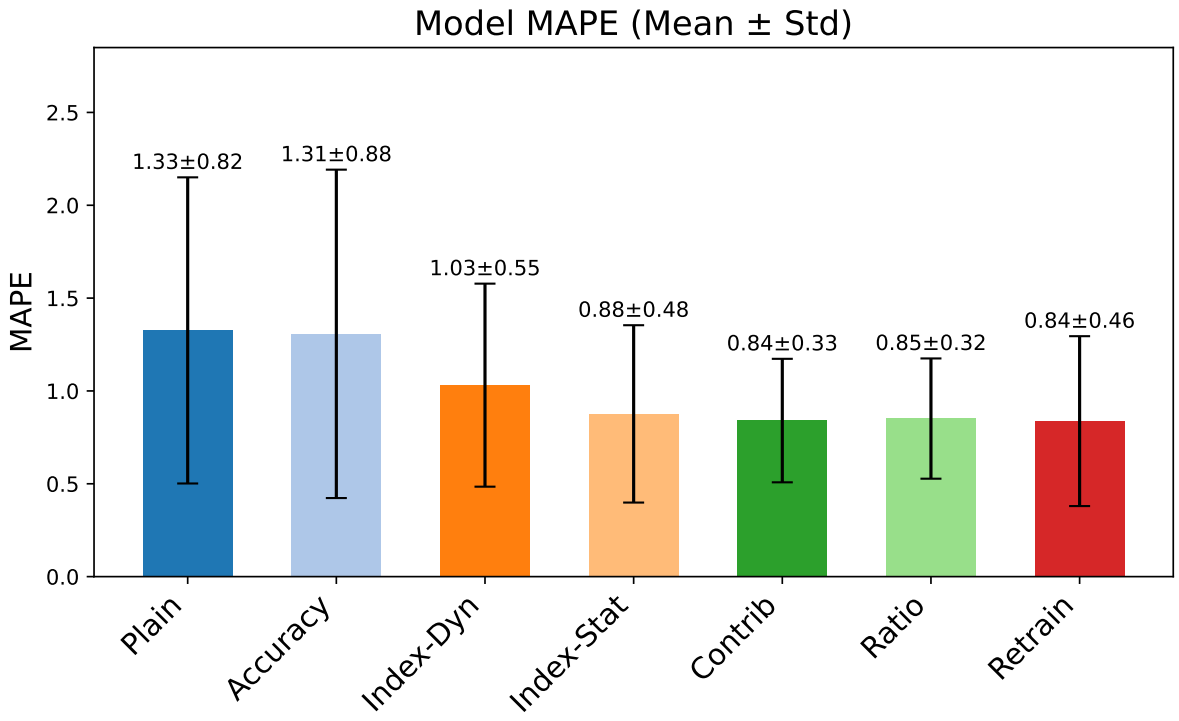


Figure 3.7: Model MAPE with Six Weeks of Training Data

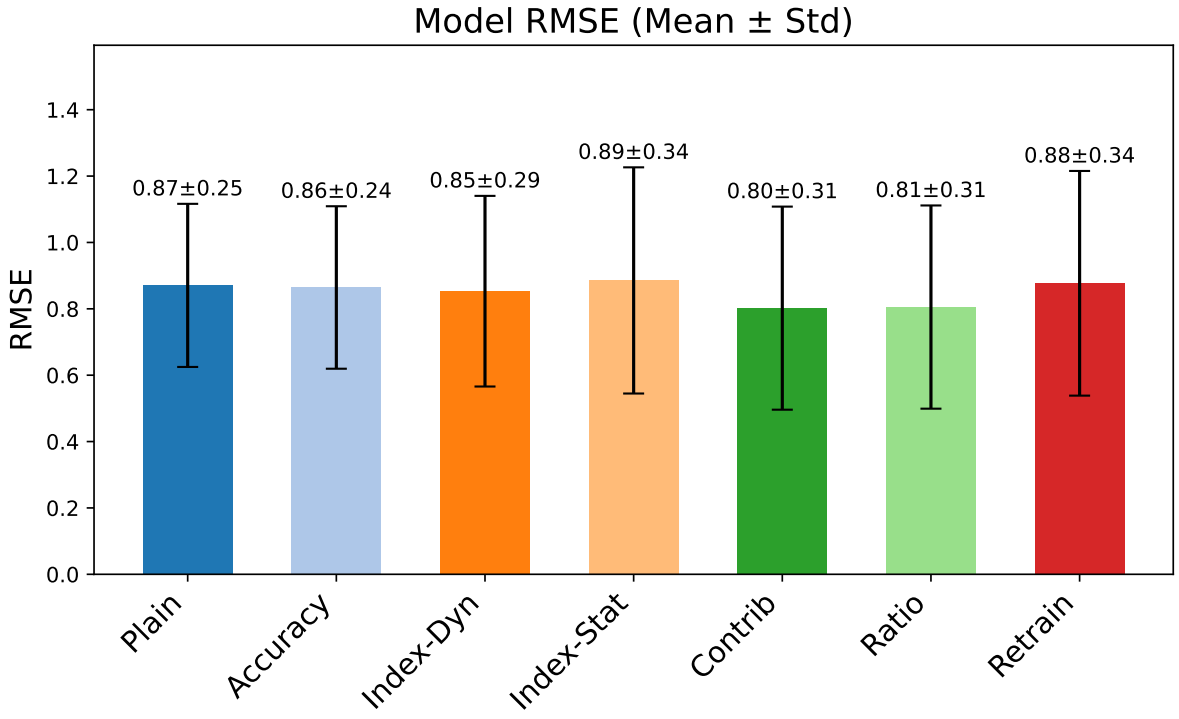


Figure 3.8: Model RMSE with Six Weeks of Training Data

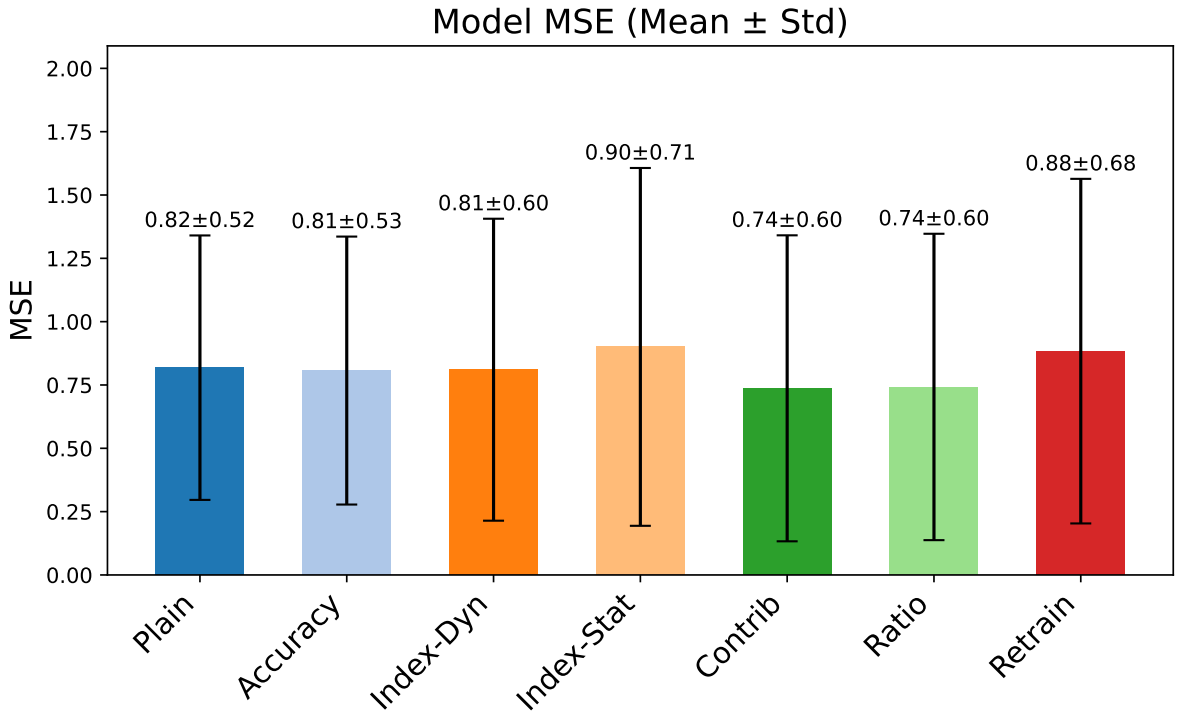


Figure 3.9: Model MSE with Six Weeks of Training Data

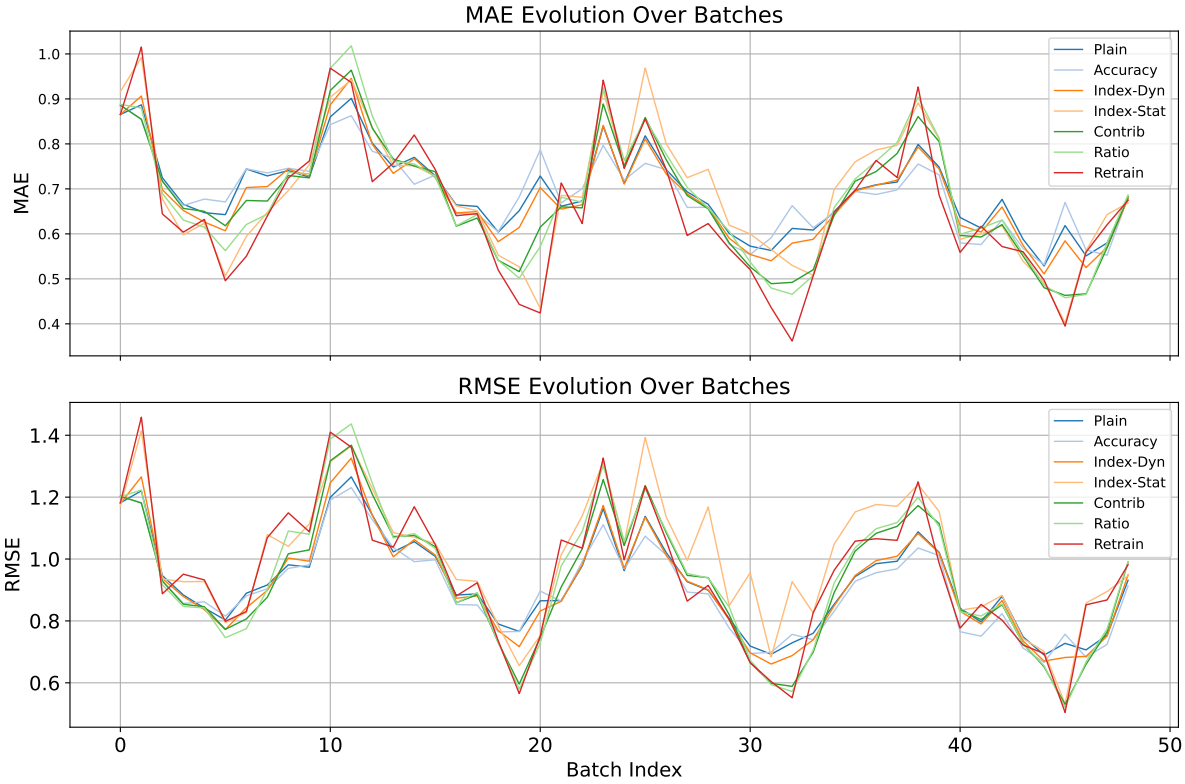


Figure 3.10: Model Error Evolution For 6 Weeks Training, 4 Week Forecast Horizon Setup

Table 3.1: Six Weeks of Training, One Day Forecast Horizon

| Model | AvgTime(s) | MAE | MSE | RMSE | MAPE |
|---|---|---|---|---|---|
| Plain | **0.0376** | 0.6847 | 0.8183 | 0.8706 | 1.3260 |
| Accuracy | 0.2086 | 0.6813 | 0.8069 | 0.8643 | 1.3075 |
| Index-Dyn | 0.2081 | 0.6373 | 0.8103 | 0.8531 | 1.0311 |
| Index-Stat | 0.4324 | 0.6359 | 0.9003 | 0.8856 | 0.8764 |
| Contrib | 0.1935 | **0.5898** | **0.7369** | **0.8020** | 0.8404 |
| Ratio | 0.1807 | 0.5943 | 0.7422 | 0.8053 | 0.8512 |
| Retrain | 0.0523 | 0.6279 | 0.8836 | 0.8769 | **0.8374** |

Table 3.2: Six Weeks of Training, Four Weeks Forecast Horizon

| Model | AvgTime(s) | MAE | MSE | RMSE | MAPE |
|---|---|---|---|---|---|
| Plain | 1.4664 | 0.6948 | 0.8708 | 0.9213 | 1.2122 |
| Accuracy | 8.2112 | 0.6890 | **0.8379** | **0.9037** | 1.2637 |
| Index-Dyn | 8.1394 | 0.6844 | 0.8665 | 0.9161 | 1.1309 |
| Index-Stat | 14.8051 | 0.6877 | 1.0414 | 1.0006 | 0.8368 |
| Contrib | 6.9280 | 0.6771 | 0.9023 | 0.9275 | 0.9933 |
| Ratio | 6.6706 | 0.6803 | 0.9296 | 0.9389 | 0.9345 |
| Retrain | **1.3902** | **0.6588** | 0.9543 | 0.9525 | **0.7904** |

Table 3.3: One Year of Training, One Week Forecast Horizon

| Model | AvgTime(s) | MAE | MSE | RMSE | MAPE |
|---|---|---|---|---|---|
| Plain | **0.3103** | 0.6919 | 1.1328 | 1.0325 | 0.5077 |
| Accuracy | 3.1599 | 0.6929 | 1.1349 | 1.0335 | 0.5089 |
| Index-Dyn | 3.1603 | 0.6684 | 0.9466 | 0.9470 | 0.8999 |
| Index-Stat | 2.6423 | **0.6333** | **0.8304** | **0.8888** | 0.8249 |
| Contrib | 4.2390 | 0.6665 | 1.0692 | 1.0014 | **0.5071** |
| Ratio | 4.2243 | 0.6677 | 1.0655 | 0.9980 | 0.5147 |
| Retrain | 0.6020 | 0.6434 | 0.8589 | 0.9032 | 0.8367 |

Table 3.4: Average Metrics Across Three Setups

| Model | AvgTime(s) | MAE | MSE | RMSE | MAPE |
|---|---|---|---|---|---|
| Plain | **0.6048** | 0.6905 | 0.9406 | 0.9415 | 1.0153 |
| Accuracy | 3.8599 | 0.6877 | 0.9266 | 0.9338 | 1.0267 |
| Index-Dyn | 3.8359 | 0.6634 | **0.8745** | **0.9054** | 1.0206 |
| Index-Stat | 5.9599 | 0.6523 | 0.9240 | 0.9250 | 0.8460 |
| Contrib | 3.7868 | 0.6445 | 0.9028 | 0.9103 | 0.7803 |
| Ratio | 3.6919 | 0.6474 | 0.9124 | 0.9141 | **0.7668** |
| Retrain | 0.6815 | **0.6434** | 0.8989 | 0.9109 | 0.8215 |

Table 3.5: Average Model Ranks Across Three Setups

| Model | Time | MAE | MSE | RMSE | MAPE | All Error Metrics |
|---|---|---|---|---|---|---|
| Plain | **1.0** | 7.0 | 7.0 | 7.0 | 5.0 | 6.5 |
| Accuracy | 6.0 | 6.0 | 6.0 | 6.0 | 7.0 | 6.2 |
| Index-Dyn | 5.0 | 5.0 | **1.0** | **1.0** | 6.0 | 3.2 |
| Index-Stat | 7.0 | 4.0 | 5.0 | 5.0 | 4.0 | 4.5 |
| Contrib | 4.0 | 2.0 | 3.0 | 2.0 | 2.0 | **2.2** |
| Ratio | 3.0 | 3.0 | 4.0 | 4.0 | **1.0** | 3.0 |
| Retrain | 2.0 | **1.0** | 2.0 | 3.0 | 3.0 | **2.2** |

# Chapter 4

# Discussion and Limitations

This section discusses the findings, limitations and avenues for future research. The main empirical finding of this research is none of the tested models consistently outperformed the rest. The only exception is that the Plain model was consistently the fastest model; though in the second setup it was very tiny bit slower than the Retrain model (see Table **??**). While there is no ready-made explanation for this, it needs to be stated that the recursive forecast likely took more time than the training. Our implementation of it is not as optimized as the scikit-learn's implementation of the fit() method. On the other hand, the Retrain model undertook both recursive forecasting and retraining.

In terms of accuracy, contribution-based eGBDT and the Retrain models are the best. The two versions of cbeGBDT calculate the $K$ trees to prune differently, but the Contrib model is leading scoreboard along Retrain model while Ratio model is right behind them (see Table 3.4 and Table 3.5).

Although Plain model is the worst in Table 3.4, its performance is not as bad is it was expected. This might indicate that the concept drift in the data is of Source I (see Figure 2.1). Although the seasons change and the model does not know that, the relationship between $X$ and $Y$ remains the same: $P_t(Y \mid X) = P_{t+\Delta}(Y \mid X)$. The lower or higher lags inform the model enough so that it knows it should predict lower values in summer or higher values in winter. Nevertheless, the relationship between lags and the predicted variable is not wildly changing over time.

However, these overall average metrics obscure the fact that in different setups different models were the most accurate. In the third setup (Table 3.3), the classic index-based elastic GBDT is decisively the most accurate model while in the second setup (Table 3.2) Accuracy is performing quite well. All in all, the results show that no single model outperforms the rest in all setups. Rather, choosing a model for deployment largely depends on the setup.

Training size, batch size, and forecast horizon length all have major impact on model's performance. The first experimental setup imitated edge device limitations and instituted the sensible forecast horizon for energy distributor application of one day. The imitation led to a very small training dataset size compared to test dataset size. The training dataset comprised of 3% of the data while the test dataset was the remaining 97%. While such large test size provided space for incremental learning to really change the ensemble, it arguably advantaged the incrementally learning models which thus could learn on more data. Yet, the Plain model performed somewhat well. This may be due to overfitting of the incremental learning models on the relatively short forecast horizon of one day. This seems to be confirmed by the fact that Retrain model and index-based eGBDT models performed the worst. These three models are the most flexible and therefore overfitting models. Hence, the fact that no single model leads the scoreboard should be both ascribed to the different setups. But which setup will lead to the success of which model is not clear from this research.

Another issue is the question of hyperparameter tuning vs. additional training. Perhaps, the Plain model benefited from getting larger training dataset more than the incremental learning models benefited from choosing their tuning paramaters. For example, the Accuracy model tended to chose the least flexible hyperparameters and so kept itself the most similar to the base GBDT model. This might have yielded the best performance during the three-week long hyperparameter training, but might have led to not flexible enough model during the test dataset. In contrast, the Plain model could have changed radically with three extra weeks of data. Thus it could have benefited more from the three weeks than the incremental learning models benefited from the tuning.

We can see that less flexible models have more volatile error on a batch than more flexible models. Accuracy chose the least flexible hyperparameters and Plain did not change during the batch learning at all. These two models consequently have lower errors when more flexible models have high errors (see Figure 3.10). However, the same applies during batches when more flexible models such as when Retrain, Index-Stat or Index-Dyn had their most accurately predicted batches. The less flexible models in such situations did not manage to adapt to the "easily" predictable batch. It seems that Retrain, Index-Stat or Index-Dyn are the most flexible

models, Contrib and Ratio are somewhat flaxible and Accuracy with its tuned parameters along with Plain are the least flexible models. The flexible models tend to overfit while the least flexible models tend to have high bias.

One structural problem of the Accuracy model (Alg. 4) in its current form is that the total weight update is independent from the error on the batch of the whole ensemble. While the individual trees' weights reflect their contribution to the reduction of the ensemble's error, the same is not true for the whole ensemble. Currently, how much the weights get updated is determined by a fixed hyperparameter named "ratio_power". This by hypeparameter tuning ended up to However, some kind of mechanism which would lead to faster weight update on conceptually drifting batches and slower weight update during non-drifting periods could improve the the model's performance. Indeed, such mechanic is crucial for the relatively successful Ratio model.

A major theoretical question is the calculation of tree contribution. Already talked about in Section 3.2.1, the current implementation of tree contribution calculation uses recorded data. Thus, the algorithm can rank and subsequently prune trees which perform poorly on new conceptually drifted data. However, such such contribution calculation differs from the way the algorithm predicts next batch. During prediction the algorithm employs recursive forecasting which leads to earlier predictions being used as features for later predictions. If the contribution was calculated with recursive forecasting as well, it would better reflect the performance during real prediction. On the other hand, trees would be pruned not based on their predictions of recorded input data but based on already predicted features. Those already have some error. Hence, the pruning would be based on trees predicting outcome variables based on non-recorded somewhat erroneous features. In this experiment only recorded features were used to calculate contribution, but using recursive forecasting or a combination of recursive forecasting and real features could lead to more accurate pruning and therefore better performing models.

# Chapter 5

# Conclusion

This master thesis research aimed to improve Elastic GBDT model proposed by [6]. It thus sought answer to the following research question:

*How can the Elastic GBDT algorithm be adapted for regression tasks to limit memory use and computational resources while enhancing predictive accuracy, and ensuring compatibility across various types of decision trees?*

Three distinct algorithms which departed from Elastic GBDT were presented. The simplest Index-Dyn merely maintained a fixed number of trees and pruned all trees after the index of the best performing tree. It then retrained as many trees as it deleted. Based on the ranking (see Table 3.5) it already very slightly outperformed Elastic GBDT (named "Index-Stat" in the table).

More advanced mechanic of contribution based tree pruning was presented under the name of contribution-based elastic GBDT (for full explanation see Alg 3). CbeGBDT calculates the contribution of each tree to the reduction of error and it prunes and subsequently retrains the least contributing trees. Two approaches named "weighted contribution" and "ratio" determining how many trees should be pruned and retrained were presented. This algorithm is on par with the Retrain model which

In addition, accuracy updated elastic GBDT was presented (see Alg 4). It maintained tree contribution calculation; however it took inspiration from Learn++.NSE and AUE. It employed weighting mechanism (AUE) using activation function (Learn++.NSE). Although it performance was the worst, it has the highest number of hyperparamters to tune. Hence, it may be possible that with different hyperparameter choice and tuning length - tuning was three weeks long for all three setups -, it could yield stronger results.

The first setup of only six-weeks long training and hyperparameter tuning with one day forecast horizon imitated edge computing. Strongly performing incremental learning could be employed and limited memory and compute devices to help stabilize energy grid by accurate

energy use forecast. For this reason the research offered algorithms with non-increasing memory requirements and the setup of very short training. However, two different setups of longer training and longer forecast horizon showed that performance of models is strongly influenced by these parameters.

While the contribution calculation managed to deliver version of Elastic GBDT which has limited memory and slightly better accuracy, it needs to be further researched in recursive forecasting applications. Currently, contribution calculation differs from batch prediction once data-point prediction relies on a past data-point prediction as one of its features - that is when prediction is recursive. In contrast, tree contribution calculation takes place only once the model receives all the recorded data from the last batch. In our implementation the contribution calcuation fully relied on the recorded data. Integrating recursive forecasting into contribution calculation can lead to further performance improvement.

# Bibliography


[1] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering*, 31(12):2346–2363, 2018.

[2] Oliver Smith, Oliver Cattell, Etienne Farcot, Reuben D O'Dea, and Keith I Hopcraft. The effect of renewable energy incorporation on power grid stability and resilience. *Science advances*, 8(9):eabj6734, 2022.

[3] Ahmed Sharique Anees. Grid integration of renewable energy sources: Challenges, issues and possible solutions. In *2012 IEEE 5th India International Conference on Power Electronics (IICPE)*, pages 1–6, 2012.

[4] Oluwaseyi Oladele. Renewable energy systems and integration into the grid. 11 2024.

[5] Yehui Li, Dalin Qin, H Vincent Poor, and Yi Wang. Introducing edge intelligence to smart meters via federated split learning. *Nature Communications*, 15(1):9044, 2024.

[6] Kun Wang, Anjin Liu, Jie Lu, Guangquan Zhang, and Li Xiong. An elastic gradient boosting decision tree for concept drift learning. In *Australasian Joint Conference on Artificial Intelligence*, pages 420–432. Springer, 2020.

[7] Ruidong Chen, Tianci Dai, Yanfeng Zhang, Yukun Zhu, Xin Liu, and Erfan Zhao. Gbdt-il: Incremental learning of gradient boosting decision trees to detect botnets in internet of things. *Sensors*, 24(7):2083, 2024.

[8] Dariusz Brzeziński and Jerzy Stefanowski. Accuracy updated ensemble for data streams with concept drift. In *International conference on hybrid artificial intelligence systems*, pages 155–163. Springer, 2011.

[9] Abdulgani Kahraman, Mehmed Kantardzic, and Muhammed Kotan. Dynamic modeling with integrated concept drift detection for predicting real-time energy consumption of industrial machines. *IEEE Access*, PP:1–1, 09 2022.

[10] Dinithi Jayaratne, Daswin De Silva, Damminda Alahakoon, and Xinghuo Yu. Continuous detection of concept drift in industrial cyber-physical systems using closed loop incremental machine learning. *Discover Artificial Intelligence*, 1:1–13, 2021.

[11] Tomás Cabello-López, Manuel Cañizares-Juan, Manuel Carranza-García, Jorge Garcia-Gutiérrez, and José C Riquelme. Concept drift detection to improve time series forecasting of wind energy generation. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 133–140. Springer, 2022.

[12] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.

[13] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2):58, 2004.

[14] Fabian Hinder, Valerie Vaquet, and Barbara Hammer. One or two things we know about concept drift—a survey on monitoring in evolving environments. part a: detecting concept drift. *Frontiers in Artificial Intelligence*, 7:1330257, 2024.

[15] Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508, 2001.

[16] Firas Bayram, Bestoun S Ahmed, and Andreas Kassler. From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*, 245:108632, 2022.

[17] Jeffrey C Schlimmer and Richard H Granger. Incremental learning from noisy data. *Machine learning*, 1:317–354, 1986.

[18] Guozhu Dong, Jiawei Han, Laks VS Lakshmanan, Jian Pei, Haixun Wang, and Philip S Yu. Online mining of changes from data streams: Research problems and preliminary results. In *Proceedings of the 2003 ACM SIGMOD Workshop on Management and Processing of Data Streams*, pages 739–747, 2003.

[19] Wei Fan. Streamminer: A classifier ensemble-based engine to mine concept-drifting data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1257–1260, 2004.

[20] Marcos Salganicoff. *Explicit forgetting algorithms for memory based learning*. Univ., 1993.

[21] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23:69–101, 1996.

[22] Mihai M Lazarescu, Svetha Venkatesh, and Hung H Bui. Using multiple windows to track concept drift. *Intelligent data analysis*, 8(1):29–59, 2004.

[23] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM, 2007.

[24] Passent ElKafrawy, Hassan Moharram, and Ahmed Awad. Optimizing adwin for steady streams. 2022.

[25] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*, pages 286–295. Springer, 2004.

[26] Manuel Baena-Garcıa, José del Campo-Ávila, Raul Fidalgo, Albert Bifet, Ricard Gavalda, and Rafael Morales-Bueno. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, volume 6, pages 77–86. Citeseer, 2006.

[27] Kyosuke Nishida and Koichiro Yamauchi. Detecting concept drift using statistical testing. In *International conference on discovery science*, pages 264–269. Springer, 2007.

[28] Roberto SM Barros, Danilo RL Cabral, Paulo M Gonçalves Jr, and Silas GTC Santos. Rddm: Reactive drift detection method. *Expert Systems with Applications*, 90:344–355, 2017.

[29] Anjin Liu, Yiliao Song, Guangquan Zhang, and Jie Lu. Regional concept drift detection and density synchronized drift adaptation. In *IJCAI international joint conference on artificial intelligence*, 2017.

[30] Isvani Frias-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yailé Caballero-Mota. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, 2014.

[31] Botao Jiao, Yinan Guo, Cuie Yang, Jiayang Pu, Zhiji Zheng, and Dunwei Gong. Incremental weighted ensemble for data streams with concept drift. *IEEE Transactions on Artificial Intelligence*, 5(1):92–103, 2022.

[32] Dariusz Brzezinski and Jerzy Stefanowski. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE transactions on neural networks and learning systems*, 25(1):81–94, 2013.

[33] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106, 2001.

[34] Richard Brendon Kirkby. *Improving hoeffding trees*. PhD thesis, The University of Waikato, 2007.

[35] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

[36] Matteo Frigo, Charles E Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 285–297. IEEE, 1999.

[37] W Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382, 2001.

[38] Haixun Wang, Wei Fan, Philip S Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–235, 2003.

[39] J Kolter and M Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Int'l Conf. Data Mining (ICDM)*, 2001.

[40] Dhouha Mejri, Riadh Khanchel, and Mohamed Limam. An ensemble method for concept drift in nonstationary environment. *Journal of Statistical computation and Simulation*, 83(6):1115–1128, 2013.

[41] Parneeta Sidhu and MPS Bhatia. A two ensemble system to handle concept drifting data streams: recurring dynamic weighted majority. *International journal of machine learning and cybernetics*, 10:563–578, 2019.

[42] Ryan Elwell and Robi Polikar. Incremental learning in nonstationary environments with controlled forgetting. In *2009 international joint conference on neural networks*, pages 771–778. IEEE, 2009.

[43] Gregory Ditzler and Robi Polikar. Incremental learning of concept drift from streaming imbalanced data. *IEEE transactions on knowledge and data engineering*, 25(10):2283–2301, 2012.

[44] Symone Gomes Soares and Rui Araújo. An on-line weighted ensemble of regressor models to handle concept drifts. *Engineering Applications of Artificial Intelligence*, 37:392–406, 2015.

[45] scikit-learn developers. GradientBoostingRegressor. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html, 2025. Accessed: 2025-04-29.

[46] Yash Chauhan and Anuj Duggal. Different sorting algorithms comparison based upon the time complexity. *International Journal of Research and Analytical Reviews*, 3(3):114–121, 2020.

[47] UCI Machine Learning Repository. Individual household electric power consumption data set. https://www.kaggle.com/datasets/uciml/electric-power-consumption-data-set, 2019. Accessed: 2025-04-23.

[48] UNECE. Smart statistics from smart meter data? https://unece.org/sites/default/files/2023-03/04_SmartMeterData_Denmark_SHalifax.pdf, 2023. https://unece.org/sites/default/files/2023-03/04_SmartMeterData_Denmark_SHalifax.pdf.

[49] GoiEner. An electricity smart meter dataset of spanish households. *Scientific Data*, 2023. https://www.nature.com/articles/s41597-023-02846-0.

[50] UK Government. Smart meters: a guide for households, 2021. Accessed: 2025-05-03.

[51] Martin Pullinger, Ellen Zapata-Webborn, Jonathan Kilgour, Simon Elam, Jessica Few, Nigel Goddard, Clare Hanmer, Eoghan McKenna, Tadj Oreszczyn, and Lynda Webb. Capturing variation in daily energy demand profiles over time with cluster analysis in british homes (september 2019–august 2022). *Applied Energy*, 360:122683, 2024.

[52] Mehrnaz Anvari, Elisavet Proedrou, Benjamin Schäfer, Christian Beck, Holger Kantz, and Marc Timme. Data-driven load profiles and the dynamics of residential electricity consumption. *Nature communications*, 13(1):4593, 2022.

[53] Artem Tokarevskikh. Modeling and properties of autoregressive processes. Study Program: Informatics Knowledge Engineering, 2023. Supervised by Ing. Kamil Dedecius, Ph.D., Informatics Knowledge Engineering, Department of Applied Mathematics. Valid until the end of the summer semester 2022/2023.

[54] Baeldung. Choosing the best q and p from acf and pacf plots in arma-type models. *Baeldung*, 2023. Explains how to select AR and MA terms using ACF and PACF plots.

[55] Valentine Shkulov. Advanced techniques for time series data feature engineering. *HackerNoon*, 2023. Explores advanced feature engineering techniques for time series data, including the application of Fourier Transform for capturing frequency components.

[56] Luis Javier Herrera, Hector Pomares, Ignacio Rojas, Alberto Guillén, Alberto Prieto, and Olga Valenzuela. Recursive prediction for long term time series forecasting using advanced models. *Neurocomputing*, 70(16-18):2870–2880, 2007.

[57] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2):211–252, 1964.

[58] Jason W. Osborne. Improving your data transformations: Applying the box–cox transformation. *Practical Assessment, Research, and Evaluation*, 15:1–9, 2010.

[59] The SciPy community. *scipy.stats.boxcox*, 2025. Accessed: 2025-05-15.

[60] B Benson, WD Pan, A Prasad, GA Gary, and Q Hu. Forecasting solar cycle 25 using deep neural networks. *Solar Physics*, 295(5):65, 2020.

[61] Jason Brownlee. *Introduction to Time Series Forecasting with Python*. Machine Learning Mastery, 2017. Covers time-based feature engineering such as extracting day of week, day of month, and month.

[62] Kosala Bandara, Christoph Bergmeir, and Hansika Hewamalage. Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. In *International Conference on Neural Information Processing*, pages 462–

474. Springer, 2020. Includes discussion of calendar-based features like day-of-month in forecasting accuracy.

[63] Scikit learn developers. sklearn.model_selection.timeseriessplit, 2024. Accessed: 2025-05-04.

[64] Yusuf A. Sha'aban. Predictive models for short-term load forecasting in the uk's electrical grid. *PLoS ONE*, 19(4):e0297267, 2024.

[65] Michał Narajewski. Probabilistic forecasting of german electricity imbalance prices. *Energies*, 15(14):4976, 2022.

[66] Neda Maleki, Oxana Lundström, Arslan Musaddiq, John Jeansson, Tobias Olsson, and Fredrik Ahlgren. Future energy insights: Time-series and deep learning models for city load forecasting. *Applied Energy*, 374:124067, 2024. 24 hour forecast horizon.

[67] Mariia Bilousova, Anton Motornenko, and Fabian Hofmann. Automating storage arbitrage in german electricity market. In *Proceedings of the International Renewable Energy Storage Conference (IRES 2022)*, volume 16, page 63. Springer Nature, 2023. Studies the differences between intra-day and day-ahead energy markets forecasting.

[68] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[69] Leo Breiman, Jerome Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

# Appendices

# Chapter A

# Introduction to GBDT

Originally proposed by Friedman [68], Gradient Boost Decision Tree is a machine learning algorithm which turns number of weak learners (decision trees) into a much stronger learner by grouping them into an ensemble. Each new tree is trained to predict and thus reduce the errors made by the immediately preceding tree. The prediction is achieved by approximating the negative gradient of an arbitrary differentiable loss function. Final prediction of the whole ensemble is the sum of all the individual trees' predictions. This subsection briefly presents GBDT algorithm on top of which my incremental learning approaches are built. The presentation of GBDT is brief due to the word limit of this work and because the core of GBDT is left unchanged by the proposed incremental learning approaches.

Formally, given a training dataset $\{(x_i, y_i)\}_{i=1}^{n}$ and a differentiable loss function $\ell(y, f(x))$, GBDT constructs the final model as an additive combination of $M$ weak learners:

$$F_M(x) = \sum_{m=1}^{M} \gamma_m h_m(x), \tag{A.1}$$

where $h_m(x)$ is the $m$-th decision or regression tree (weak learner) and $\gamma_m$ is its weight (or multiplier). Classification and Regression Trees (CART) were introduced in 1984 by Breiman et. al. [69]. Due to the limits of scope their construction will not be explored in this thesis; however, since the data are numerical only regression trees are used.

At each iteration $m$, the algorithm calculates the pseudo-residuals, defined as the negative gradient of the loss function evaluated at the model from the previous iteration $F_{m-1}(x)$:

$$r_{im} = -\left[\frac{\partial \ell(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=F_{m-1}(x)}, \quad i = 1, \ldots, n. \tag{A.2}$$

A weak learner $h_m(x)$ is then fitted to these pseudo-residuals $\{(x_i, r_{im})\}_{i=1}^{n}$. Subsequently,

the multiplier $\gamma_m$ is calculated through line search as:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} \ell(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))." \tag{A.3}$$

Finally, the model is updated by adding the new weighted learner (in my case regression tree):

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x). \tag{A.4}$$

The GBDT algorithm is computationally efficient and achieves high accuracy, which makes it particularly appealing for applications constrained by computational resources and memory, such as forecasting in smart meters. GBDT remains unchanged by the incrementally learning methods presented earlier. Instead, they focus on improving the model's adaptability to evolving data distributions by weighting and/or pruning the weak learners.