Energy Efficiency in Machine Learning

Approaches to Sustainable Data Stream Mining

Eva García-Martín

Blekinge Institute of Technology thesis for the degree of doctor of philosophy series
No: 2020:02

Energy Efficiency in Machine Learning

Approaches to Sustainable Data Stream Mining

Eva García-Martín

Thesis for the degree of Doctor of Philosophy in Computer Science

Faculty of Computer Sciences
Department of Computer Science
Blekinge Institute of Technology
SWEDEN

Faculty of Computer Sciences
Department of Computer Science
Publisher: Blekinge Institute of Technology,
SE-371 79 Karlskrona, Sweden
Printed by Lenanders, Kalmar, Sweden 2020
ISBN 978-91-7295-396-3
ISSN 1653-2090

“That brain of mine is something more than merely mortal; as time will show”

Ada Lovelace

“Todo por El Humor”

Javier Espinosa

Abstract

Energy efficiency in machine learning explores how to build machine learning algorithms and models with low computational and power requirements. Although energy consumption is starting to gain interest in the field of machine learning, the majority of solutions still focus on obtaining the highest predictive accuracy, without a clear focus on sustainability.

This thesis explores green machine learning, which builds on green computing and computer architecture to design sustainable and energy-efficient machine learning algorithms. In particular, we investigate *how to design machine learning algorithms that automatically learn from streaming data in an energy-efficient manner*.

We first illustrate how energy can be measured in the context of machine learning, in the form of a literature review and a procedure to create theoretical energy models. We then use this knowledge to analyze the energy footprint of Hoeffding trees, presenting an energy model that maps the number of computations and memory accesses to the main functionalities of the algorithm. We also analyze the hardware events correlated to the execution of the algorithm, their functions and their hyper parameters.

The final contribution of the thesis is showcased by two novel extensions of Hoeffding tree algorithms, the Hoeffding tree with *nmin adaptation* and the Green Accelerated Hoeffding Tree. These solutions are able to reduce the energy consumption of the original algorithms by twenty and thirty percent, with minimal impact on accuracy. This is achieved by setting an individual splitting criteria for each branch of the decision tree, spending more energy on the fast growing branches and saving energy on the rest.

This thesis shows the importance of evaluating energy consumption when designing machine learning algorithms, proving that we can design more energy-efficient algorithms and still achieve competitive accuracy results.

Keywords: machine learning, energy efficiency, data stream mining, green machine learning, edge computing

To Matilde Benito Morteirín

Preface

The author has been the main driver of all the publications where she has been the first author. The author has planned the studies, designed the experiments, conducted the experiments, conducted the analysis and written the manuscripts.

Included Papers

PAPER I García-Martín E., Rodrigues C., Riley G., and Grahn H. (2018) *Estimation of Energy Consumption in Machine Learning.* Journal of Parallel and Distributed Computing, 134, (pp.75-88), Elsevier. DOI: `https://doi.org/10.1016/j.jpdc.2019.07.007`

PAPER II García-Martín E., Lavesson N., and Grahn H. (2017). *Energy Efficiency Analysis of the Very Fast Decision Tree algorithm.* In: Missaoui R., Abdessalem T., Latapy M. (eds) Trends in Social Network Analysis. Lecture Notes in Social Networks, (pp. 229-252), Springer. DOI: `https://doi.org/10.1007/978-3-319-53420-6_10`

PAPER III García-Martín E., Lavesson N., and Grahn H. (2017). *Identification of Energy Hotspots: A Case Study of the Very Fast Decision Tree.* In: Au M., Castiglione A., Choo KK., Palmieri F., Li KC. (eds) Green, Pervasive, and Cloud Computing. GPC 2017. Lecture Notes in Computer Science, 10232, (pp. 267-281). Springer. DOI: `https://doi.org/10.1007/978-3-319-57186-7_21`

PAPER IV García-Martín E., Lavesson N., Grahn H., Casalicchio E., and Boeva V. (2018) *Energy-Aware Very Fast Decision Tree.* Jour-

nal of Data Science and Analytics, Springer (*Accepted, to appear*).

PAPER V García-Martín E., Bifet A., and Lavesson N., (2019) *Energy Modeling of Hoeffding Tree Ensembles*. Intelligent Data Analysis. (*Accepted, to appear*)

PAPER VI García-Martín E., Bifet A., and Lavesson N., (2019) *Green Accelerated Hoeffding Tree*. Submitted to: The 24th Pacific-Asia Conference on Knowledge Discovery and Data Mining. (*Under review*).

Related Papers

PAPER VII García-Martín E., Lavesson N., and Grahn H. (2015) *Energy Efficiency in Data Stream Mining.* Advances in Social Networks Analysis and Mining (ASONAM), 2015 IEEE/ACM International Conference on. IEEE, 2015. DOI: `10.1145/2808797.2808863`

PAPER VIII García-Martín E., Lavesson N., and Doroud M. (2016). *Hashtags and followers:An experimental study of the online social network Twitter*, Social Network Analysis and Mining (SNAM), **6**(1) (pp. 1-15), Springer. DOI: `https://doi.org/10.1007/s13278-016-0320-6`

PAPER IX García-Martín E., Lavesson N., Grahn H., and Boeva V. (2017, May). *Energy Efficiency in Machine Learning: A position paper.* In 30th Annual Workshop of the Swedish Artificial Intelligence Society SAIS 2017, May 15–16, 2017, Karlskrona, Sweden 137, (pp. 68-72). Linköping University Electronic Press.

PAPER X García-Martín E., and Lavesson N. (2017). *Is it ethical to avoid error analysis?* 2017 Workshop on Fairness, Accountability, and Transparency in Machine Learning (FAT/ML 2017), held in conjunction with the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. arXiv preprint `arXiv:1706.10237`.

PAPER XI Abghari, S., García-Martín E., Johansson C., Lavesson N., and Grahn H. (2017). *Trend Analysis to Automatically Identify Heat Program Changes.* Energy Procedia, 116, (pp. 407-415).

PAPER XII Lundberg L., Lennerstad H., Boeva V., García-Martín E. (2019) *Increasing the Margin in Support Vector Machines through Hyperplane Folding.* Proceedings of the 2019 11th International Conference on Machine Learning and Computing, ACM. DOI: `https://doi.org/10.1145/3318299.3318319`

PAPER XIII García-Martín E., Lavesson N., Grahn H., Casalicchio E., and Boeva V. (2018) *Hoeffding Trees with nmin adaptation.* In: The 5th IEEE International Conference on Data Science and Advanced Analytics (DSAA 2018),(pp. 70-79), IEEE. DOI: `DOI10.1109/DSAA.2018.00017`

PAPER XIV García-Martín E., Lavesson N., Grahn H., Casalicchio E., and Boeva V. (2018) *How to Measure Energy Consumption in Machine Learning.* In: ECML-PKDD 2018 1st International Workshop on Energy Efficient Data Mining and Knowledge Discovery (Green Data Mining). ECML PKDD 2018 Workshops, vol 11329, Lectures Notes in Artificial Intelligence, Springer. DOI: `https://doi.org/10.1007/978-3-030-13453-2_20`

The work in this thesis is part of the research project *Scalable resource-efficient systems for big data analytics* funded by the Knowledge Foundation (KKS)(grant: 20140032).

Acknowledgements

This thesis would not have existed without the guidance, support, and trust from Niklas Lavesson. I am forever grateful for having someone so kind and encouraging helping me navigate this journey. Thanks for being not only a remarkable advisor, but also an exceptional friend. Special thanks to Håkan Grahn, for having the patience to believe in me when we both knew there was room for improvement. To Veselka Boeva and Emiliano Casalicchio, thanks for your help in this adventure. Truly warm gratitude to Albert Bifet, gràcies. Thanks for welcoming me at the lab in Paris, for all the nice discussions even from the other side of the world that have made this thesis shine. Thanks for being you, always smiling, always so close, always so nice. To Darja Šmite, for her constant support, confidence, and inspiration.

To those colleagues that are now very good friends. Shahrooz Abghari, Christian Nordahl, Diego Navarro, Thomas Sievert. Thanks for making BTH somewhere worth going to. To my friends, my family, who never left my side. Ane M Martín, Javier Espinosa, Amaia Oleaga, Antonio Zaragoza, Loreto Fernández, Vanesa Jerez, María J. Zato, Patricia Zazo, Bárbara González, Jenny Valenzuela, Bárbara Gil, Caroline Meliones, Laura Martínez, Ely Bosquez, Evelin Fuster, Bea López, Alejandro Sanz. To the new people that are now part of my life, Annaam Butt, Pooja Sousthanmath, Raquel Santos.

To my favorite person, who is always lighting the path, Marta Díaz. To my family, who make everything worth it. Felisa Martín, Ángel García, Elisa Candelori, Matilde García, Javier Blasco, Pablo Blasco, Sonia De Frutos, Álvaro Blasco, Juan Martín, Elena Martín, María L. Larios, Juan P. Martín, Maribel García, Javier Sánchez, Adrian Sánchez, Mari Martín, Guy Pabion, Sylvie Pabion, Ruben Díaz, David Pabion, Paloma Del Pozo, Bea Toribio, Juan García, Borja García. To Matilde B. Morteirín, para ti, siempre.

Göteborg, October 2019

Contents

1 Introduction

Machine learning is a core sub-area of artificial intelligence, which provides computers the ability to automatically learn from experience without being explicitly programmed for it [1]. Machine learning models are present in many current applications and platforms. For example, speech recognition at Google [2–4], image recognition at Facebook [5, 6], and movie recommendations at Netflix [7, 8]. Data stream mining is a subfield of machine learning that investigates how to process potentially infinite streams of data that change with time [9, 10]. Algorithms in this field have the ability to update the model in real time with the arrival of new and evolving data, and by reading the data only once [11].

Energy efficiency and power consumption in hardware have been highly researched for decades in the area of computer architecture [12]. This interest has also started to become present in machine learning. Researchers, especially in the fields of machine learning systems [13] and edge computing [14], have realized that energy-efficient algorithms are needed to move machine learning algorithms to the device [15]. This is desirable in scenarios where there is no possibility to send the data to the cloud, and for privacy concerns so that the data does not leave the user device.

This thesis investigates *resource-efficient machine learning*, with the goal to design and develop sustainable and energy-efficient machine learning algorithms. In particular, we focus on energy efficiency in data stream mining. Algorithms in this domain are designed to run constantly on embedded devices. Although these algorithms are known to have a small memory and energy footprint [16], reducing their energy consumption can have a significant impact on the device and battery performance. Throughout this thesis, energy efficiency is defined as the energy consumed for a given task. Thus, a reduction in energy consumption is translated to a higher

energy efficiency.

The thesis is divided into two parts, comprising six papers. The first part of the thesis addresses the challenge of how to measure energy consumption in machine learning algorithms. We present a survey on energy estimation methods from the computer architecture field, synthesized for the machine learning audience. The second part of the thesis investigates how to reduce the energy consumption of Hoeffding trees, by presenting, with different levels of maturity, solutions and new energy-efficient Hoeffding tree algorithms.

1.1 Research Problem

The aim of this thesis is to investigate *how can one design machine learning algorithms that automatically learn from streaming data in an energy-efficient manner.* To address this aim, we focus on the following objectives:

1. *Investigate how to measure energy consumption in machine learning algorithms.* This objective is fulfilled with the first and fifth studies of the thesis, where we present empirical and theoretical approaches to estimate energy consumption and its applicability in machine learning.

2. *Investigate how to make Hoeffding tree algorithms more energy efficient.* To address this objective we explore, with different levels of maturity, how Hoeffding tree algorithms consume energy and how to create energy-efficient Hoeffding tree algorithms. We first identify the energy bottlenecks and energy consumption patterns of the Very Fast Decision Tree (VFDT) algorithm [16], the original Hoeffding tree algorithm, by showing which parameter setups and functions consume the highest amount of energy. We then present the *nmin adaptation* method for Hoeffding trees and ensembles of Hoeffding trees to reduce their energy consumption. We finally introduce the Green Accelerated Hoeffding Tree algorithm, an extension of the latest Hoeffding tree that reduces its energy consumption while maintaining the same predictive accuracy.

To fulfill those objectives, we present the following research questions, answered in the upcoming chapters:

RQ1. *How can energy be measured in the context of machine learning algorithms and applications?*
Papers I [17] and V address this question by presenting, first, practical approaches to estimate energy consumption, and second, a generic approach to theoretically estimate energy consumption of machine learning algorithms.

RQ2. *What are the energy consumption patterns of Hoeffding trees?*
Papers II [18], III [19], and IV address this question by presenting, first, the energy footprint of Hoeffding trees when varying the different parameters. Second, the most energy consuming functions of Hoeffding trees. Third and final, a theoretical energy model that shows where and how energy is consumed in Hoeffding trees.

RQ3. *How to improve existing Hoeffding tree algorithms in terms of energy consumption?*
Papers IV, and V address this question by presenting the *nmin adaptation* method for Hoeffding trees and ensembles of Hoeffding trees. This method adapts the *nmin* parameter, responsible for the main energy hotspot of Hoeffding trees, to reduce their energy consumption while marginally affecting accuracy.

RQ4. *How can we further improve predictive accuracy of Hoeffding Trees while still being within certain energy constraints?*
Paper VI addresses this question by presenting an extension of Hoeffding trees which significantly improves accuracy while being energy efficient. This is achieved by adaptively growing the Hoeffding tree depending on the characteristics of the data and the different branches.

Figure 1.1 gives an overview of the aforementioned papers, and their relation to the aim and objectives described above.

1.2 Contributions

The main contribution focuses on how to build, develop, and design energy-efficient data stream mining algorithms. In particular, we present the following contributions, achieved through the papers presented in this thesis.

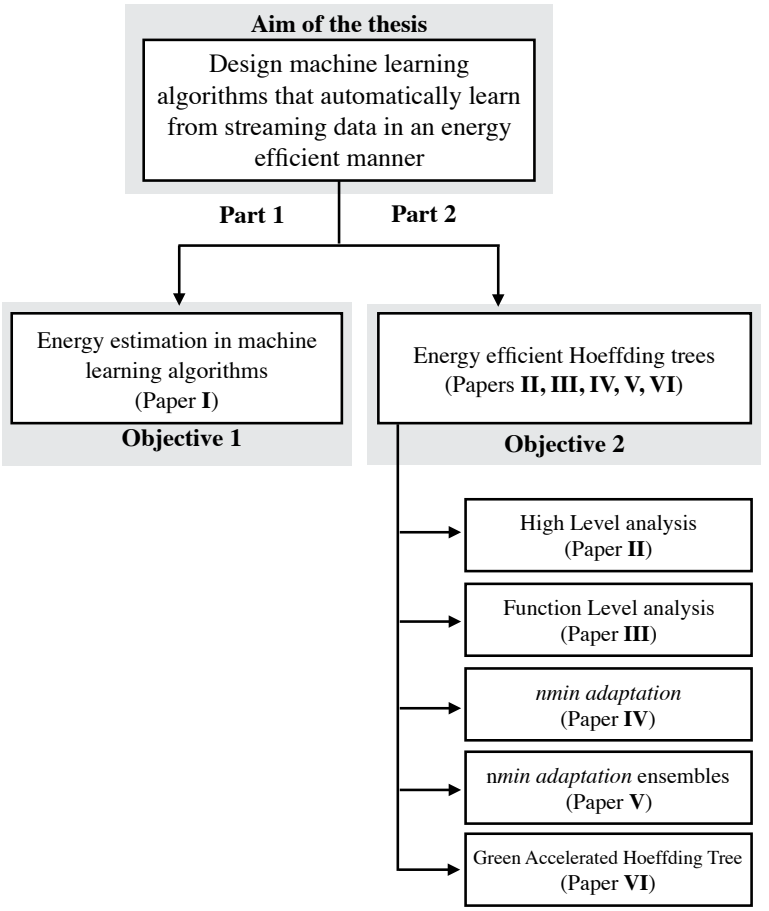

Figure 1.1: *Papers included in the thesis*

Energy Consumption in Machine Learning Energy consumption has been widely studied in the field of computer architecture for decades. However, although there is some recent interest to build machine learning models with a small energy footprint [14], still there is a lack of literature that explains how energy can be estimated in a machine learning context. To address this gap, this thesis first presents guidelines on how to empirically estimate energy consumption, taken from the computer architecture field and mapped to machine learning scenarios. We also detail existing tools to measure energy consumption straightforward in the computer, and the state-of-the-art approaches in machine learning to estimate energy consumption.

Second, we present a theoretical approach to estimate energy consumption for any class of algorithm (Figure 4.1). We apply this knowledge to showcase detailed theoretical energy models for Hoeffding trees and ensembles of online decision trees for concept drift scenarios. These energy models aim at giving further insight and understanding of how an algorithm consumes energy independently of the hardware platform.

Energy Efficient analysis of Hoeffding trees Papers II, III, and IV present an analysis of how Hoeffding trees consume energy. Apart from the theoretical energy model already mentioned in the first contribution, we present how this class of algorithms consume energy for different parameters, functions, and features. We first present the impact of the different parameter setups on energy consumption. We then present a sensitivity analysis on the number of instances, numerical and nominal attributes, and their effect on energy consumption. We finally present which functions of the algorithms are the most energy consuming. We discovered interesting patterns such as the high impact on energy consumption of handling numerical attributes, and of calculating the attributes that better split the data. Finally, Paper VI presents a unique measure, called energy efficiency, that shows how much energy is needed to achieve a higher accuracy. This shows that less energy is needed to train on the first 80 percent of instances compared to the last 20 percent of instances. This is relevant for streams of data where the end users might have a tight energy budget, allowing them to stop training after reaching a set accuracy threshold.

Energy Efficient Splitting Criteria of Hoeffding trees Traditional Hoeffding trees grow the decision tree following the same splitting criteria for each leaf. Papers IV, V, and VI present a novel approach to build Hoeffding trees in a more energy-efficient manner, by having an ad-hoc splitting criteria for each branch/leaf. The *nmin adaptation* method, presented in Papers IV, and V, sets a unique value of the *nmin* parameter for each leaf, so that only the necessary energy is spent on that branch. This method adapts the value of the batch size of instances needed to be observed at each leaf to make a confident split, to avoid the case where the best attributes are calculated but then a split did not occur because the leaf did not observe enough instances. This method is applied to standard Hoeffidng trees (Paper IV) and ensembles of Hoeffding trees (Paper V). Paper VI presents the Green

Accelerated Hoeffding Tree (GAHT). This algorithm chooses a splitting criteria depending on the fraction of instances that a particular node has observed so far. A less restrictive splitting criteria is set to those leaves that have observed significantly more instances than the average leaf, increasing accuracy and energy consumption. On the other hand, energy is saved by deactivating the less visited nodes, which consequently contribute less to an accuracy increase. These methods have shown to reduce the energy consumption significantly (more than 20 percent) while maintaining the same levels of accuracy as its competitor (Extremely Fast Decision Tree).

1.3 Outline

The remainder of the thesis is divided into eleven chapters. Chapter 2 explains the necessary background to understand the main concepts of the thesis. It follows a top-down approach, starting from a more general view on machine learning, to a more detailed view in data stream mining, online decision trees and Hoeffding trees. We conclude with a discussion about how energy is consumed by software programs.

Chapter 3 gives an overview of the scientific method used to conduct the studies. We introduce computer science and machine learning, formulate the research questions and the experiment design, datasets explanation, and data analysis. Chapter 4 presents the results of the thesis. We synthesize the contributions of the papers and show their relationship with the aim and objective presented in this section. Finally, Chapter 5 concludes with a summary and synthesis of the contributions and main points of the thesis. The remainder, Chapters 6 - 11, are Papers I-VI included in the thesis.

2 Background

2.1 Machine Learning

Machine learning has its foundations in artificial intelligence, computer science, and statistics. In 1950 [20], Alan Turing proposed the Turing test, a test to determine if a machine could deceive the interrogator to believe that responses to specific questions came from a human person rather than from a machine. Two years later, Arthur Samuel created the first game-playing program for checkers [1] and introduced the term *machine learning*. In 1956, *artificial intelligence* first appeared as a term, with the idea to build intelligent entities [21]. In 1957, the first perceptron algorithm was created [22]. A few years later, machine learning started to gain importance after having a more specific focus on statistics.

Mitchell [23] provided the following definition for learning: *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.* In the following subsections we provide different examples of tasks T, experiences E, and performance measures P, inspired on the work by Goodfellow et al [24].

2.1.1 The Task T

There are many different kind of tasks in machine learning, from object recognition, to machine translation. The most common tasks that are covered in these thesis are classification, regression, and anomaly detection. Many other tasks exist, but are out of the scope of this background explanation [24].

Classification tasks require for the program to predict a categorical value given some input. The input data is usually written in the form of

a vector x, and the output is identified by y. The goal is to find function $f : \mathbb{R}^n \rightarrow \{1, ..., k\}$, that maps the input to one of the categories k, where y is representing one of those k categories. Thus, y being the result of applying function f to the input, in the form of $y = f(x)$.

Regression tasks require for the program to predict a numerical value. The goal is to output a function that given some real input x, outputs another real number y, represented by: $f : \mathbb{R}^n \rightarrow \mathbb{R}$.

Anomaly Detection tasks focus on detecting instances that are atypical or rare, usually called outliers. A very typical example is intrusion detection, shown in the CICIDS [25] dataset on Section 3.3.

There are many tasks that are not covered in this chapter, such as machine translation. Detailed explanations of them are available on Chapter 5.1.1 of *Deep Learning* [24].

2.1.2 The Performance P

Performance is defined differently in different fields. In this thesis, performance refers to predictive performance, that is, how many correctly classified instances did the model output, i.e. accuracy. It is calculated as the proportion of instances, out of all instances, that are correctly classified. Another common used metric is error rate, which is the proportion of instances, out of all instances, that are incorrectly classified. That is, one minus the accuracy.

To evaluate the accuracy or error rate the dataset is divided into training and test sets. The training set is used to produce the model, and the test set to evaluate the model's performance. Thus the algorithm's performance would be evaluated on how well it works on unobserved data. In the data stream mining scenario the complete dataset is not available. The performance on these scenarios is evaluated by first testing the model on an instance, and then using that instance for training. The accuracy values are then calculated as the model is trained, on pseudo real-time. This is called prequential evaluation [26].

2.1.3 The Experience E

Machine learning algorithms are broadly divided into supervised, unsupervised, or reinforcement learning, depending on the type of dataset that they

experience.

Supervised learning cover the set of algorithms that have the complete dataset as input, knowing what is the correct output. That is, the inputs x, mapped with the output y is known. These algorithms then learn from the correct output and full examples to create the machine learning model. Classification and regression tasks are part of supervised learning.

Unsupervised learning have the dataset available, but the answer with the correct output is missing. For example a dataset of music songs. The goal for these algorithms is to learn properties of the structure of the dataset. Examples are clustering algorithms, where instances are clustered into groups of similar instances based on their feature vectors.

Reinforcement learning cover a different set of algorithms, which interact with the environment. Everytime the algorithm performs an action in an environment, it gets some feedback, in the form of rewards. Based on the feedback it chooses the next action, to achieve an already set goal. A classical example of reinforcement learning is an algorithm learning to play a game, e.g. chess.

2.2 Data Mining

Data mining is usually referred to as knowledge discovery from databases. The goal is to find patterns in data, and that could be through machine learning algorithms, e.g. decision trees or neural networks, or through statistical analyses, e.g. correlation analysis. It can be considered as applied machine learning, where the goal is the specific application, such as large scale analytics, rather than developing a new learning algorithm.

There are many different fields and disciplines in data mining, such as parallel and distributed data mining, big data analytics, graph analysis, recommender systems, data stream mining, etc. This thesis has a specific focus in data stream mining, which we explain more in detail in Section 2.3.

2.3 Data Stream Mining

Data streams are a sequence of instances, potentially infinite, for real-time analytics [11]. The instances, which arrive one by one, are used to create

machine learning models on real time, without the need to store all the data. The main requirements of data stream mining algorithms are the following [11]:

1. Process each instance at a time, inspecting it at most once.
2. Resource efficient: use a limited amount of memory and time.
3. Ready to predict at any time.
4. Concept drift handling: adapt and handle temporal changes.

A data stream mining classifier typically follows five steps: i) get an unlabeled instance x, ii) make a prediction $\hat{y}$ based on the current model f, iii) get the true label y for that instance x, iv) train and update the model f based on the pair (x, y), v) update the models statistics and performance by comparing $\hat{y}$ and y.

There are also other classes of tasks, such as regression, clustering (unsupervised learning), and frequent pattern mining. More details are given in the *Machine Learning for Data Streams with Practical Examples in MOA* book [11]. In this thesis we focus on classification of data streams using decision trees, which are explained in detailed in the next section.

Data streams evolve over time, due to a *shift in the statistical properties of the data, more than what can be attributed to chance fluctuations* [11]. This is known as *concept drift*, and it can be divided into abrupt or gradual, and can affect the complete instance space or a part of it. Novel data stream mining algorithms, such as the Hoeffding Adaptive Tree [27], are able to adapt the model when concept drift is detected. It is important to notice that change in the data stream is different from outliers and noise. There are different change detection methods, such as `ADWIN` [27], CUMSUM test [28], and the drift detection method (DDM) [29].

Applications of data streams are many, such as sensor data and Internet Of Things (IoT), social media, and health care.

2.4 Hoeffding Trees

Hoeffding trees are a type of decision trees for data streams. They are built incrementally, from potentially infinite streams of data, by saving a few statistics at each node.

A common decision tree example is shown in Figure 2.1. Offline (standard) decision trees follow a divide and conquer approach that divides the input space into local regions, that are identified in a sequence of recursive splits in a smaller number of steps [30]. Unlike online decision trees (decision trees for data streams), all the data is available, so the split is made on the attribute with the highest entropy.

Hoeffding trees use the Hoeffding bound [31] (Eq.2.1) to decide how many instances (n) have to be observed at a node to make a split with confidence $1 - \delta$.

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{2.1}$$

where R is the range of the random variable, 1-δ the desired probability, and n the number of examples seen at a node.

The theorem states that if the difference in information gain between the best (A) and the second best attribute (B) is higher than the Hoeffding bound ϵ ($(IG(S, A) - IG(S, B)) > \epsilon$), then n instances are enough to make a confident split on attribute A. It has been shown [16] that the output of the Hoeffding tree is asymptotically nearly identical to the batch decision tree, if infinite number of examples would be available.

Information gain (IG) is the entropy caused by partitioning the examples according to that attribute [23], and is defined as [23]:

$$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.2}$$

where S_v is the partition for which the attribute has value v and $Values(A)$ is the set of all possible values of the attribute A [23]. Information gain represents how much information is gained after partitioning the set with a specific attribute. Thus, the goal is to obtain the attribute which is able to partition the dataset, with the current observed instances, giving the highest

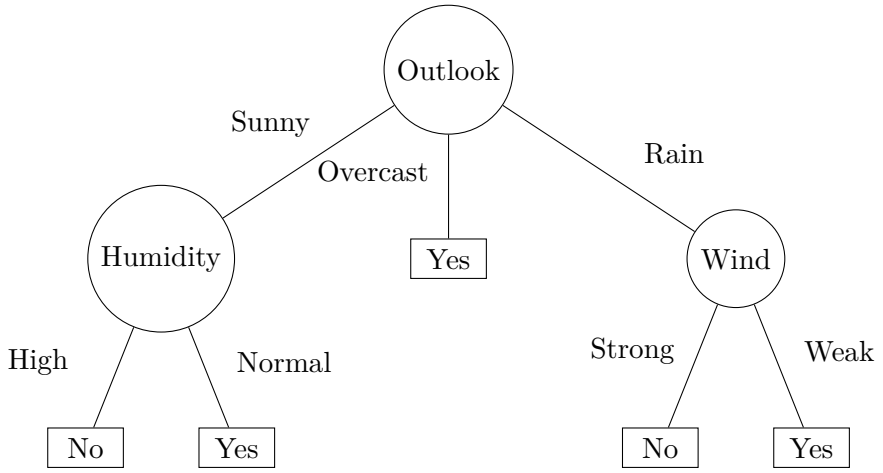

Figure 2.1: *Standard decision tree example.*

information. The entropy of instances S is defined as [23]:

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2 p_i \tag{2.3}$$

where p_i represents the proportion of instances that belong to class i. Entropy varies between zero and one. *Zero* means that all instances belong to the same class, thus no information is needed to classify such instances. *One* means that the instances contain a variation of the class values, thus there is a lot of information needed to predict the class that they belong to [23].

Hoeffding trees are built as the data arrives, by storing the necessary information at each node or leaf to calculate the information gain of the different attributes to obtain a confident split. Statistics about nominal attributes are stored in the form of a table, with every pair of {class,attribute value} stored at each cell. Storing numerical attributes is expensive, in terms of energy and memory consumption (Chapter 4). There are different ways to store the statistics about numerical attributes. Currently, the Gaussian distribution, with the mean, standard deviation, maximum and minimum values is stored for each {attribute,class} pair.

The following sections present the different Hoeffding tree algorithms used in this thesis.

2.4.1 VFDT

The Very Fast Decision Tree, presented in Alg. 1, is the implementation of the Hoeffding tree with the following modifications:

Algorithm 1 VFDT: Very Fast Decision Tree

```
1: HT: Tree with a single leaf (the root)
2: X: set of attributes
3: G(·): split evaluation function
4: τ: hyperparameter set by the user
5: while stream is not empty do
6:    Read instance I_i
7:    Sort I_i to corresponding leaf l using HT
8:    Update statistics at leaf l
9:    Increment n_l: instances seen at l
10:   if nmin ≤ n_l then
11:      Compute Ḡ_l(X_i) for each attribute X_i
12:      X_a, X_b = attributes with the highest Ḡ_l
13:      ΔḠ = Ḡ_l(X_a) − Ḡ_l(X_b)
14:      Compute ε using Eq. 2.1
15:      if (ΔḠ > ε) or (ε < τ) then
16:         Replace l with a node that splits on X_a
17:         for each branch of the split do
18:            Set new leaf l_m with initialized statistics
19:         end for
20:      else
21:         Disable attr {X_p|(Ḡ_l(X_p) − Ḡ_l(X_a)) > ε}
22:      end if
23:   end if
24: end while
25: end while
```

- *nmin* parameter: the *nmin* parameter controls the minimum number of instances that should be observed at a node before calculating the best attributes. This saves energy by calculating the information gain for a batch of *nmin* instances, rather than for every new instance. Chapter 9 extends this further by proposing an adapting value of *nmin* for each node, depending on the characteristics of the data that has been observed.

- τ parameter: if the information gain of the two best attributes is very similar, the tree would wait for a very long time until a split is made, since that difference needs to be bigger than ϵ. To avoid this, the τ parameter was introduced, so that when $IG(A) - IG(B) < \epsilon < \tau$, a split is made.
- Deactivating less promising nodes to be more resource efficient.

2.4.2 Concept-Adapting Very Fast Decision Tree

The Concept-Adapting Very Fast Decision Tree (CVFDT) [32] was introduced as an extension of the VFDT to handle concept drift scenarios. The CVFDT grows an alternate branch when, after re-checking the splits, the algorithm has observed that it would have split on a different attribute with the most updated instances. If the alternate branch is performing better than the original branch, the alternate branch replaces the original branch, otherwise, the alternate branch is removed. This is checked periodically. In order to perform these comparisons, the CVFDT keeps a sliding window of the latest instances at the different nodes, instead of just the necessary statistics at the leaves like for the VFDT. This requires more resources, in terms of memory and energy consumption.

The rest of the algorithm follows the same procedure as the VFDT (Alg. 1). That is, whenever a new instance arrives, that instance traverses the tree until the corresponding leaf is reached, updating the statistics at the leaf. Whenever *nmin* examples are observed at a leaf, the best attributes are calculated to check for a possible split.

2.4.3 Hoeffding Adaptive Tree

The Hoeffding Adaptive Tree (HAT) [27] algorithm in an extension of the Hoeffding tree algorithm that could handle concept drift with theoretical guarantees. HAT uses the `ADWIN` [27, 33] change detector to detect for concept drift.

`ADWIN` is the state-of-the-art change detector for data streams. It keeps a window with the recently seen instances, ensuring that the window has the maximal length being consistent with the hypothesis: *there has been no change in the average value inside the window.* Values are then added and dropped to keep the hypothesis true. The algorithm compares all

possible subwindows, checking if they exhibit *distinct enough* averages. If that occurs, a change is detected, and the instances in the older subwindow are dropped [11].

The HAT algorithm grows alternate branches the moment a change is detected using `ADWIN`. The old branches are replaced by the new ones as soon as it detects that the new branch outputs higher predictive performance. To keep track of the changes the algorithm keeps track of the error between the predicted and the real values of the stream. This introduces significant overhead compared to the original VFDT, but achieves significantly higher accuracy in concept drift datasets.

2.4.4 Extremely Fast Decision Tree

The Extremely Fast Decision Tree (EFDT) [34] was introduced as an extension to the VFDT that would converge faster to a higher accuracy. This extension has two main new characteristics:

- Split criteria: EFDT introduces a less restrictive split criteria on the leaf. Instead of comparing the information gain of the two best attributes against the Hoeffding Bound ϵ, it compares the information gain of a null split against the best attribute. That is, if the information gain of the best attribute is higher than the information gain of no splitting (calculated as the information gain of the statistics at that leaf), by a difference of ϵ, then a split occurs. This creates faster splits creating a more accurate tree with less instances.
- Split re-evaluation: Since the new split criteria can lead to non-optimal splits, the algorithm re-evaluates the splits every certain number of instances.

This algorithm has shown high accuracy results, at a cost of a higher energy consumption. Chapter 11 (Green Accelerated Hoeffding Tree algorithm) extends this algorithm to make it more energy efficient.

2.5 Ensembles

Ensembles are combinations of machine learning models to form a final, more accurate model. The predictions of the smaller models are combined

in different ways, for example through averaging, or voting [11, 35]. We describe *bagging, boosting, and accuracy-weighted* in this section, and refer to [35] for more thorough explanations on the other techniques.

Bagging consists of creating M different models that are trained on different samples of the data. Each sample is chosen randomly with replacement, i.e. the same sample can potentially be used to train more than one of the M models. The final model makes a prediction by taking the majority voted class between the M models. Online Bagging was introduced to create ensembles of algorithms for data streams. More details are given in Section 2.5.1.

Accuracy-weighted ensembles are designed to combine the predictions of the different classifiers by assigning a specific weight to each classifier. The stream is analyzed in the form of chunks. After every chunk of data is analyzed, a new classifier is built, which substitutes the less performing classifier. This makes the classifier adapt to changes in the incoming data. The performance of each classifier is evaluated by calculating the predictive error on the most recent data chunk. The Online Accuracy Updated Ensemble (OAUE) algorithm builds on this technique in Section 2.5.3.

Boosting is a technique that combines multiple models sequentially instead of in parallel. The main idea is that the new model is built based on the performance of the previous model, setting a higher weight to the examples that are misclassified by the previous models [36]. This increases performance by better generalizing throughout the data. Boosting was later extended for streaming and online scenarios [37, 38].

2.5.1 Online Bagging

Online Bagging [37, 38](Alg. 2) is the extension of the bagging technique for online and streaming scenarios. Since the examples are read only once, drawing examples randomly with replacement poses a big challenge. To address that challenge, this algorithm reads each instance assigning it a specific weight following a Poisson distribution with $\lambda = 1$, for each model. After each instance is read, each model will be trained on the new instance k times, where $k = Poisson(1)$.

The rationale behind using a Poisson distribution lies in the property that a *boostrap sampling can be simulated by creating K copies of each instance*

Algorithm 2 Online Bagging(h,d)

```
1: for each instance d do
2:    for each model h_m, (m ∈ 1, 2, ..., M) do
3:       k = Poisson(λ = 1)
4:       Train the model on the new instance k times
5:    end for
6: end for
```

following a binomial distribution [11]. When the number of instances is large, the binomial distribution tends to a Poisson distribution with $\lambda = 1$.

2.5.2 Leveraging Bagging

Leveraging Bagging [39] enhances Online Bagging by proposing two randomization improvements: increasing resampling and increasing output randomization. Instead of setting $\lambda = 1$, λ is set to a hyperparameter chosen by the user, increasing resampling. Output detection codes are used to increase randomization at the output of the ensemble, which changes the way majority voting is performed to increase diversity between classifiers.

On top of that, Leveraging Bagging uses `ADWIN` [40] to adapt to concept drift. When a change is detected, the algorithm removes the worst classifier of the ensemble, and a new classifier is added.

2.5.3 Online Accuracy Updated Ensemble

The Online Accuracy Updated Ensemble (OAUE) [41] is an algorithm that builds on the weighted-accuracy technique for online scenarios. It presents two main new strategies:

- Sliding windows: The batch version of the OAUE, namely Accuracy Updated Ensemble, uses a fixed chunk size that needs to be estimated by the user or an expert. The OAUE uses a sliding window with the latest d examples to update the weight of all classifiers. The weight is calculated based on the prediction error of such classifier based on the sliding window of the latest d examples. This faster adapts to changes.
- Weight: To ensure that the latest data points have more importance than the old ones, the new classifiers are trained on the newest examples

and given the highest weight of the old classifiers.

This algorithm follows the following approach. First, for each instance, it calculates the predictive error of all classifiers, and all classifiers are trained on the new instance. If d instances have been observed, the weight of all classifiers is updated substituting new classifiers for old ones.

2.5.4 Online Boosting

Online Boosting (Alg. 3) was the first extension of the boosting technique for streaming and online scenarios [37, 38]. It is similar to Online Bagging.

Algorithm 3 Online Boosting(h,d)

```
1: λ_d = 1
2: for each instance d do
3:    for each model h_m, (m ∈ 1, 2, ..., M) do
4:       k = Poisson(λ_d)
5:       Train the model on the new instance k times
6:       if instance is correctly classified then
7:          instance weight λ_d updated to a lower value
8:          instance weight λ_d updated to a higher value
9:       end if
10:   end for
11: end for
```

The only difference is that the λ of the Poisson distribution will be updated based on the correctly or incorrectly classification of the previous classifier in the ensemble. After an instance is read, the first model is trained on that instance k times, where $k = Poisson(\lambda = 1)$. If the model classifies correctly that instance, the weight of that instance gets updated to a lower value. That instance is then observed by the next model, with the λ updated to the new weighted value. If the instance is misclassified, the weight of that instance increases, so that the next model is trained more times on that instance.

2.5.5 Online Coordinate Boosting

Online Coordinate Boosting [42] was introduced to better approximate Online Boosting to AdaBoost [36], the original batch boosting algorithm.

The procedure is similar to Online Boosting (Section 2.5.4), differing only in the weight update procedure. They derive the weight value by minimizing AdaBoost's loss when viewed in an incremental form. More details are given in the original paper [42].

2.6 Energy Efficiency

Energy efficiency is the key concept in this thesis, and it can have several different definitions in different areas. For this thesis, energy efficiency is defined as the energy consumed for a given task. We use the term as means of comparing several different algorithms in terms of their energy consumption. Thus, if algorithm a is more energy efficient than algorithm b, algorithm a consumes less energy than algorithm b.

We also define energy efficiency as *energy proportionality* in Chapter 11. In that context, energy efficiency refers to the proportion of energy spent to achieve a certain accuracy. The ideal scenario occurs when a higher energy consumption results in a higher accuracy, since the energy spent has a positive effect on accuracy. In the streaming scenario it often occurs that there is a high increase of accuracy consuming a certain amount of energy for the first instances, and then that same amount of energy is needed for a few more percentages of energy. Thus, the algorithm is more energy efficient at the beginning of the execution and less at the end, being less proportional. This is clearly observed in Figure 4.8 in Chapter 4.

2.6.1 Motivation

The key reason why energy consumption and energy efficiency are studied in this thesis is due to the high impact on energy consumption of machine learning algorithms. A recent study [43] has quantified the enviornmental costs, in terms of CO_2 emissions, of training several neural network models commonly used in the field of natural language processing. The results show that training the transformer neural network [44] emits five times more CO_2 than one car in its lifetime. This paper brings the attention to the massive computational resources that are needed for today's machine learning tasks.

Energy consumption is also related to embedded devices and internet of things (IoT). Currently most of the training and inference of machine learning algorithms are performed in the cloud. This is because the models

are too big to fit into a small device, and because they consume too much energy. Designing low-power machine learning models allows for inference in the mobile device, which has a positive impact on privacy, user experience (due to a faster speed), and accessibility in areas where internet connection is poor. Streaming algorithms are a class of algorithms that are meant to run in embedded devices. That is the reason for our objective to build greener streaming algorithms. Both for the environmental impact and the ability to run these algorithms in the edge.

2.6.2 Measuring Energy Consumption

Measuring energy consumption is a challenging task, due to the many variables involved. Chapter 6 presents a survey of the different approaches to estimate and measure energy consumption, linked to machine learning. This section presents the definitions of energy and power consumption, as a background introduction of how programs consume energy.

Energy consumption is defined as the amount of power consumed during an interval of time [45]:

$$E = \int_0^T P(t)dt \tag{2.4}$$

Thus, **power** is defined as the rate at which energy is being consumed:

$$P_{avg} = \frac{E}{T} \tag{2.5}$$

where P_{avg} is the average power in an interval of time T, and E the energy consumption. The instantaneous power $P(t)$ consumed or supplied by a circuit element is [46]:

$$P(t) = I(t) \cdot V(t) \tag{2.6}$$

where $I(t)$ is the current and $V(t)$ is the voltage.

Power can be divided into static and dynamic power. **Static power**, known as leakage power, is defined as the power consumed when there is no circuit activity. **Dynamic power** is defined as the power dissipated by the circuit, by charging and discharging the capacitor [45].

$$P_{dynamic} = \alpha \cdot C \cdot V_{dd}^2 \cdot f \tag{2.7}$$

where α is the activity factor, V_{dd} is the voltage, C the capacitance and f the clock frequency. The activity factor shows how much part of the circuit is active. If a circuit is turned off completely, the activity factor would be zero [46]. Dynamic power has a direct effect on the energy consumption of the processor. When a processor executes a program, depending on the resources needed by that program the values of α, C, V_{dd}, and f will vary. Nowadays all modern processors use dynamic frequency voltage scaling (DVFS) to reduce the voltage and frequency values when the programs don't need that many resources. This leads to many energy savings.

A program consumes energy based on the amount of computations and memory accesses required to run such program. More specifically, the energy consumed by a program is the product of the IC (amount of instructions), the CPI (clock per instruction) and the energy per clock cycle (EPC).

$$E = IC \cdot CPI \cdot EPC \tag{2.8}$$

EPC is defined as

$$EPC \propto C \cdot V_{dd}^2 \tag{2.9}$$

Energy per instruction (EPI) is defined as the product between CPI and EPC, $EPI = CPI \cdot EPC$ [45]. Different instructions, such as floating point operations, memory accesses, have different CPI values. Horowitz et al. [47] have presented a very interesting study where they quantify the amount of energy consumed per type of instruction. The results show that DRAM accesses are 1,000 times more energy consuming that floating point operations. The reason is that there are more cycles involved in memory accesses, and there is a delay to access the memory.

We believe that the most straightforward approach to have an overview of the energy consumed of a machine learning algorithm is by having a theoretical understanding of where energy is spent, similar to the time complexity analysis of algorithms. That is the reason why we present the approach to create theoretical energy models from Section 4.1 as one of the key contributions of this thesis. The energy model shows how to estimate energy from a theoretical perspective, independent of the programming language and hardware platform.

3

Scientific Approach

This thesis is connected to the areas of machine learning and computer engineering, overlapping the areas of data stream mining and energy efficiency in software. The two foundational questions of machine learning are the following [48]: i) How can one construct computer systems that automatically improve through experience?, and ii) What are the fundamental statistical-computational-information-theoretic laws that govern all learning systems, including computers, humans, and organizations?

Data stream mining addresses how to mine potentially infinite evolving streams of data, in one pass, and incurring in low memory requirements [9]. Computer architecture is an engineering or applied science discipline that focuses on designing a computer to maximize performance while staying within cost, power, and availability constraints [49]. One such focus lies on energy and power efficiency [45]. Techniques from computer architecture are useful to design stream mining algorithm with low computational requirements.

The fundamental question of this thesis, which addresses the aforementioned questions from machine learning, data stream mining, and computer architecture, is the following: *How can one design machine learning algorithms that automatically learn from evolving streaming data in an energy-efficient manner?* The following sections detail the research methods, datasets, data analysis, energy estimation, and validity threats, that together with the research questions proposed in Section 1.1 answer the thesis fundamental question.

3.1 Research Method

In order to answer the research questions from Sections 1.1, we follow a set of quantitative and qualitative methods that are described in this section.

Paper I follows a qualitative approach in the form of a survey or literature review. The selection of articles was done based on a previous survey on computer architecture and energy estimation methods [50]. We further use that information to connect it to energy estimation in machine learning, by showing how to apply that methodology on the application of two machine learning use cases.

Papers II-VI follow a quantitative approach in the form of experiments. Table 3.1 summarizes the experimental design from Papers II-VI. The goal in those experiments is to measure the energy consumption and accuracy (dependent variables), when testing either different algorithms or different algorithm designs and setups (independent variables). Papers IV and VI use statistical tests on the data, to examine if the difference in energy and accuracy between the compared algorithms is statistically significant.

Table 3.1: *Experimental design for Papers II-VI. VFDT: Very Fast Decision Tree [16]. CVFDT: Concept-Adaptive Very Fast Deicion Tree [32]. VFDT-nmin: Very Fast Decision Tree with nmin adaptation [51]. GAHT: Green Accelerated Hoeffding Tree (Paper VI). EFDT: Extremely Fast Decision Tree [34]*

Paper	Algorithms	Dependent Variables	Independent Variables	Experiment Type
II	VFDT	Accuracy, Energy, Power	15 Parameter Setups	Parameter tuning
III	VFDT	Accuracy, Total Energy, Energy per function	14 Parameter Setups	Function analysis
IV	VFDT, VFDT-nmin, CVFDT	Accuracy, Energy	# of instances, # of numeric attributes, # of nominal attributes	Sensitivity analysis, statistical significance
V	Leveraging Bagging [39], Online Coordinate boosting [42], Online Accuracy Updated Ensemble [52], Online Bagging [38], and Online Boosting [38]	Accuracy, Energy	*nmin adaptation*	Performance evaluation
VI	VFDT, GAHT, EFDT	Accuracy, Energy, # Nodes, #Leaves,	Algorithms	Performance evaluation

3.2 Statistical Significance

Paper IV use hypothesis testing to determine if the difference in accuracy and energy consumption of the algorithms is statistically significant.

In order to decide between using parametric and non-parametric tests, we first test for normality between the differences in accuracy and the differences in energy consumption [53]. Suppose A1 and A2 are two algorithms that must be evaluated. Then, the following hypotheses are proposed:

$\boldsymbol{H_0}$: The differences in accuracy between A_1 and A_2 come from a normal distribution.

$\boldsymbol{H_1}$: The differences in accuracy between A_1 and A_2 do not come from a normal distribution

$\boldsymbol{H_0}$: The differences in energy consumption between A_1 and A_2 come from a normal distribution

$\boldsymbol{H_1}$: The differences in energy consumption between A_1 and A_2 do not come from a normal distribution

We perform a Shapiro-Wilk test for normality [54] on the energy consumption and accuracy data. If the p-value of the test is lower than 0.01, the chosen alpha level, the null hypothesis is rejected and we conclude that the data is likely not normally distributed. On the other hand, if the p-value is higher than 0.01, the null hypothesis can not be rejected, indicating that the data is normally distributed.

If the data is normally distributed we conduct a parametric test on the data, namely the paired student t-test [55]. On the other hand, if the data is not normally distributed, we conduct a non-parametric test, namely the Wilcoxon signed-rank test [56]. The proposed hypotheses are the following:

$\boldsymbol{H_0}$: $\mu_{A_{A1}} = \mu_{A_{A2}}$, where $\mu_{A_{A1}}$ and $\mu_{A_{A2}}$ represent the mean of accuracy values for $A1$ and $A2$. Thus, the null hypothesis states that the means of the accuracy values between $A1$ and $A2$ are equal.

$\boldsymbol{H_1}$: $\mu_{A_{A1}} > \mu_{A_{A2}}$, stating that the mean of the accuracy of $A1$ is higher than the mean of the accuracy of $A2$.

$\boldsymbol{H_0} : \mu_{E_{A1}} = \mu_{E_{A2}}$, where $\mu_{E_{A1}}$ and $\mu_{E_{A2}}$ represent the mean of energy consumption values for $A1$ and $A2$, respectively. Thus, the null hypothesis states that the means of the energy consumption values between $A1$ and $A2$ are equal.

$\boldsymbol{H_1} : \mu_{E_{A1}} > \mu_{E_{A2}}$, stating that the mean of the energy consumption of $A1$ is higher than the mean of the energy consumption of $A2$.

If the p-value of the test is lower than 0.01, the null hypothesis is rejected, supporting the alternative hypothesis. These are generic hypotheses on algorithms A1 and A2, that are later materialized into specific algorithms in Paper IV, adapting the hypotheses to the actual data.

3.3 Datasets

This section describes the datasets used in Papers II-VI. These datasets are used to train and test the machine learning model. Papers II and III use the same dataset for training and testing, which introduces clear limitations regarding the results. Papers IV, V, and VI overcome this limitation by training and testing on different sets of the dataset. In particular, Paper IV use 2/3 of the dataset for training, and 1/3 for testing. Papers V and VI use *prequential evaluation*, where the model first tests on a set of instances that are later used for training.

In order to increase generalizability, we use both synthetic and real world datasets. The synthetic datasets allow us to investigate the relationship between different variables by varying the number of attributes and instances. For instance, Paper IV investigates how increasing the number of instances, and number of nominal and numerical attributes affect the energy consumed by the algorithm. The real world datasets allow us to generalize the findings to real world setups, outside of the controlled environment of a synthetic dataset.

The datasets from all papers are commonly used by researchers in the stream mining field [57]. Table 3.2 shows a description of such datasets, detailing the number of attributes, classes, and types of attributes.

In particular, we use the following synthetic generators obtained from the massive online analysis (MOA) framework [58]: random tree, hyperplane,

LED, waveform, random radial basis function (RBF), and Agrawal [10]. The random tree generator builds a tree, inspired on the dataset proposed by the VFDT original authors [16], with random attributes, attribute values, and class to predict as the leaf. Attributes are generated and labeled following the path of the tree and the branches. The hyperplane generator creates a dataset following Eq. 3.1.

$$\sum_{i=1}^{d} w_i x_i = w_0, \tag{3.1}$$

where x_i is the coordinate for each point, w_i the weight for each point, and w_0 the threshold to label the points either positive or negative. More details are given in [32]. This dataset is used as a benchmark dataset with concept drift, to test how good the algorithm adapts to changes. The LED generator creates a dataset with predictions for digits on a LED display, with a total of 24 attributes. The waveform dataset creates numerical values for a total of 21 attributes that represent the coordinates of three different types of waves. Another dataset used for testing concept drift is the random RBF (Radial Basis Function) generator, which creates centroids with random center, class label and weight. Each new example selects a center, which represents the class of such example. The centroids are moving with a certain speed introducing drift in the dataset. Finally, the Agrawal [59] generator creates ten different loan functions, and the goal is to predict if the function belongs to either Group A or Group B, based on attributes such as *salary, loan*, and *age*.

The real world datasets from Papers II, III, IV, V, and VI are: poker, electricity, airline, CICIDS, forest, and KDD cup. Each instance of the poker dataset represents a poker hand, consisting of five playing cards, represented by two attributes. Each pair of attributes stands for the suit and the rank of the card. The target is to learn the kind of hand that those cards represent. The class is a numerical value between 0 and 9, 0 representing that there is no hand at the moment, 9 representing that there is a royal flush. The electricity dataset was originally described in [60], presenting some instances from the Australian New South Wales Electricity Market. The target is to predict if the electricity price goes up or down based on different attributes. The airline dataset was created by Ikonomovska [61] to predict whether a flight will be delayed or not based on the airport of origin, destination, airline,

and other attributes. The CICIDS [25] and KDD cup[1] [62] are intrusion detection datasets. The forest dataset is obtained from the US Forest Service (USFS) [63][2]. The goal is to predict the cover type (the predominant kind of tree) using only cartographic features, such as *elevation, soil type*, and *aspect*.

Table 3.2: *Datasets.* A_i*=Number of nominal attributes.* A_f*=Number of numerical attributes*

Dataset	A_i	A_f	Class	Used in
Random Tree	5	5	2	Papers II, III, IV, V, VI
Hyperplane	0	10	2	Papers II, III, VI
LED	24	0	10	Paper III, IV, V, VI
Waveform	0	21	3	Papers III, IV, V, VI
RBF	0	10	2	Papers IV, V, VI
Agrawal	3	6	2	Paper V, VI
Poker Hand	5	5	10	Papers II, IV, V, VI
Electricity	1	6	2	Paper IV, V, VI
Airline	4	3	2	Papers II, IV, V, VI
CICIDS	78	5	6	Papers IV, V, VI
Forest	40	10	7	Papers IV, V, VI
KDD Cup	7	34	23	Paper IV

3.4 Energy Estimation Tools

Energy estimation of software programs and algorithms is a challenging task. This section details the approaches used on each paper to estimate energy consumption. With the publication of the different papers we have gathered the knowledge on the best ways to estimate energy for different scenarios. While this aspect is thoroughly covered in Paper I (Chapter 6), this Section summarizes those findings. Table 3.3 synthesizes the tools, in which papers they are used, and their advantages and disadvantages.

Paper II uses the PowerAPI [64] tool to measure the energy consumption on the different scenarios. PowerAPI is a tool that estimates the energy consumed by a program based on its CPU utilization. The main limitation is that this tool does not consider accesses to DRAM, thus many energy consumption details are missing. We then use the tool Jalen for Paper III, that uses the same models as PowerAPI, but is meant for java programs.

[1] `http://kdd.ics.uci.edu/databases/kddcup99/task.html`
[2] `https://www.kaggle.com/c/forest-cover-type-prediction`

Table 3.3: *Energy Estimation Software*

Tool	Paper	Type	Advantages	Disadvantages
PowerAPI [64]	II	CPU utilization	Easy to use. No overhead	Does not consider memory accesses
Jalen [64]	III	CPU Utilization	Easy to use. No overhead	Does not consider memory accesses
Sniper [65] + McPAT [66]	IV	Simulation	High accurate models	Big overhead
Intel Power Gadget[3]	V, VI	Performance Counters	Easy to use, accurate results	Results not available per process

The motivation to use this tool is that Jalen is able to output the energy consumption per function, fulfilling the requirements of this study. This allows us to identify the energy bottlenecks of the VFDT algorithm. However, the same disadvantage occurs as with PowerAPI.

In Paper IV we use the Sniper simulator [4] to address the disadvantages of having inaccurate energy models to estimate the energy consumption. Sniper [65] is a simulator that together with McPAT [66] outputs where in the processor the energy is consumed, and how much energy is spent on accessing the DRAM and the different caches. It gives a detailed view of the energy consumption. We can also inject `SimMarker()` function calls around each function of interest in the code to obtain the energy consumption for each function. The key drawback with Sniper is that simulating a simple algorithm run consumes a lot of processor time. That was the reason why we used small datasets to be able to simulate the VFDT under different scenarios. `SimMarker()` functions incur also in high processing time.

Papers V and VI use Intel Power Gadget[5]. This tool uses Intel's RAPL interface [67], which estimates the energy consumed by the DRAM and the processor based on accesses to the performance counters on the processor. Although Intel has not open sourced the RAPL interface and energy models, further studies have validated its accuracy [68, 69]. The advantages of this approach are many. It does not introduce any overhead, and it is able to output real-time energy estimations, which is needed for data stream mining scenarios. The main disadvantage is that the energy consumption values are not isolated for a specific program. However, we have conducted our experiments in a closed environment with only the basic programs running

[4] `http://snipersim.org`

[5] `https://software.intel.com/en-us/articles/intel-power-gadget-20`

on the computer to address that limitation.

3.5 Validity Threats

This section discusses internal validity, external validity, and construct validity.

Internal Validity

Internal validity refers to inferences about whether an observed correlation between groups reflects a casual relationship [70]. A high internal validity indicates that the relationship between the independent and the dependent variable is strong with a high confidence. This indicates that no confounding variable is affecting the dependent variable.

Paper I does not pose any cause-effect claim, since it is a review of existing literature. Papers II-VI aim is to find casual links between algorithms/parameter choices and resulting accuracies and energy consumptions. However, since some experiments cannot easily be designed to clearly fulfil the requirements on experiment designs that can offer strong support on casual links, we focus on finding strong correlation links instead.

External Validity

External validity addresses the generalizability of the results [71]. Paper I covers a broad set of papers, and emphasizes its generalizability by providing two machine learning use cases from different areas showing the applicability of the proposed methodology.

Papers II-VI are concerned with either exploratory analysis of a specific algorithm (Papers II and III), or with a new method or algorithm. Paper IV presents a method, the *nmin adaptation*, that reduces the energy consumption of the VFDT. To improve generalizability of such method on other algorithms, Paper V validates its predictive performance and low energy consumption on a new set of ensembles of decision trees. Papers II-VI aim at increasing external validity by conducting the experiments on different synthetic and real world datasets from different application areas. Paper IV performs statistical significance results on the data, which improves the generalizability of the results.

Construct Validity

Construct validity addresses if a test measures what it claims [70]. It addresses how inadequate definition of variables can lead to incorrect experimental results. Our studies have variables that have a clear definition in their field. For example, energy consumption, power consumption, and accuracy. These variables have been defined in Chapter 2.

4

Results

This thesis addresses the question on *How can one design machine learning algorithms that automatically learn from streaming data in an energy-efficient manner?* Papers I-VI investigate the different aspects involved in answering the foundational question, summarized in the following steps:

1. Understand how the energy consumed by machine learning algorithms can be measured by applying the existing solutions from computer architecture.
2. Analyze the energy consumption patterns of the given algorithm. This can be done empirically and theoretically. Theoretically by creating an energy model as shown in Section 4.1. Empirically by doing exploratory analyses of parameter tuning and function analysis and their effect on energy consumption.
3. Once there is an understanding of how the algorithm consumes energy, the next step is to present algorithm extensions that are more energy efficient, focusing on the most energy consuming parts.
4. Our proposal to address such challenge is to spend the energy budget smartly. That is, using more energy only on the necessary parts of the model, and saving energy on the other parts that don't have a positive effect on accuracy.

To investigate more specifically those four steps, we proposed four research questions (Section 1.1). Papers I-VI answer those questions as follows:

RQ1. *How can energy be measured in the context of machine learning algorithms and applications?*

Paper I presents a survey on different methods to measure energy consumption. From this survey, apart from the software tools to estimate energy presented in Table 3.3 (Section 3.4), we discovered that the best approach to estimate energy in the context of data streams is to use PMCs (performance monitoring counters). This method allows for real time estimation of machine learning algorithms running under large-scale datasets. The main drawback of PMCs is that the results are not reported per process, but for the complete processor and DRAM.

Paper V shows how to construct theoretical energy models by representing an algorithm's functions in terms of number of computations and number of memory accesses. These functions are then mapped to number of attributes, number of instances, and type of attributes. We can then estimate the energy consumed for that algorithm for different dataset types, and by knowing the energy consumption of each type of operation and access, which is available in the literature [47]. More details are given in Section 4.1 and in Papers I and V.

RQ2. *What are the energy consumption patterns of Hoeffding trees?* Papers II, II, and IV address this question by presenting the energy consumption patterns of Hoeffding trees through parameter tuning, function analysis, and through a theoretical energy model. We discover that the most energy consuming part of Hoeffding trees is when splits occur. Every *nmin* instances the tree calculates the information gain for each attribute, choosing the attribute that obtains a higher information gain if that attribute would split on the current data. That is very energy consuming because many computations and memory accesses are involved in calculating the information gain for each attribute. Paper IV shows the high energy impact of increasing the number of attributes. We also conclude that numerical attributes are significantly more energy consuming than nominal attributes. The reason, taken from the empirical evaluation and theoretical energy model on Paper IV, is that more information needs to be stored and accessed to keep track of numerical attributes. At the same time handling numerical attributes involves floating point operations, which are more energy consuming than integer

operations (nominal attributes). These expensive computations were done every *nmin* instances. We discover that if those *nmin* instances were not enough to create a split, those computations would have been done unnecessarily. To increase energy efficiency in this respect, we present an adaptive value of *nmin* based on the observed data. More details are given in Section 4.2.

RQ3. *How to improve existing Hoeffding tree algorithms in terms of energy consumption?*
The *nmin adaptation* method from Papers IV and V reduces the energy consumption of standard Hoeffding trees by 30 percent, and of ensembles of Hoeffding trees by 20 percent, without significantly affecting accuracy. When a split does not occur on a specific node, the reason is that the difference in information gain between the two best attributes is either not big enough or not small enough to make a confident split. When that occurs, the *nmin adaptation* method calculates the minimum number of instances needed for a split, and adapts the *nmin* value to that number of instances. This way, we have delayed the calculations to obtain the best attributes, saving significant amounts of energy. More details are given in Section 4.3.

RQ4. *How can we further improve predictive accuracy of Hoeffding Trees while still being within certain energy constraints?*
Paper VI presents the Green Hoeffding Accelerated Tree (GAHT) algorithm, as an energy-efficient extension of the Extremely Fast Decision Tree (EFDT). We achieve an energy reduction of 20 percent, with no real impact on accuracy. The GAHT sets the less restrictive splitting criteria from the EFDT only to a subset of branches, while deactivating another subset of branches. This results in competitive accuracy results and lower energy consumption than the EFDT. More details are given in Section 4.3.

4.1 Energy Consumption in Machine Learning

Measuring energy consumption in the context of machine learning is not straightforward. Several approaches already exist that give an estimate of how much energy is consumed addressed mainly for deep learning [15, 72].

Paper I presents a survey of the different approaches to estimate energy taken from the computer architecture area, mapped to possible machine learning scenarios. In particular, we have identified four categories, which are not mutually exclusive: *PMC (Performance monitoring counters), simulation, instruction-level, architecture-level, real time.* **PMCs** are a set of special-purpose registers in modern processors that count specific event types that are hardware related. These events are also used for estimating the energy consumption of the processor and the DRAM. The main advantages are that it is application independent and that it introduces almost no overhead. Techniques in this category are very useful for estimating the energy consumption of any machine learning model, big or small. **Simulation**, on the other hand, introduces significant overhead, making it more useful for training small datasets. The main advantage of this technique is the detailed information it provides, showing where and how energy is consumed. **Instruction-level** measurements give insight into which instructions are the most energy consuming of a program. This is useful for instance to evaluate the energy consumption of different algorithmic parts, e.g. layers in a neural network. **Architecture-level** measurements are useful to design and build specific purpose hardware for machine learning [72]. Finally, **real-time** energy estimation techniques, such as PMCs, are useful in scenarios where the data is available in real time, such as data streams. Paper I gives more details on these techniques, how to use them, and examples on how to use them in different machine learning scenarios, e.g. stream mining and deep learning.

Paper V presents a theoretical approach to estimate the energy consumed by any machine learning algorithm. The goal is to represent any algorithm in terms of number of computations and number of memory accesses. Figure 4.1 shows the three steps involved. First, we create the main blocks of the structure of the energy model, which are integer (INT) operations, floating point operations (FPU), cache accesses, and cache misses. Any researcher who is interested in using this approach can either take this structure as a baseline for their energy model, or adapt it to consider more architectural blocks. The second step is to identify the main functions of the algorithm, this can be done directly from the pseudocode of the algorithm. Once this functions are identified, they can be represented in terms of number of instances, number of attributes, etc. This will be dependent on the type of algorithm. For example, if we are analyzing the function of traversing

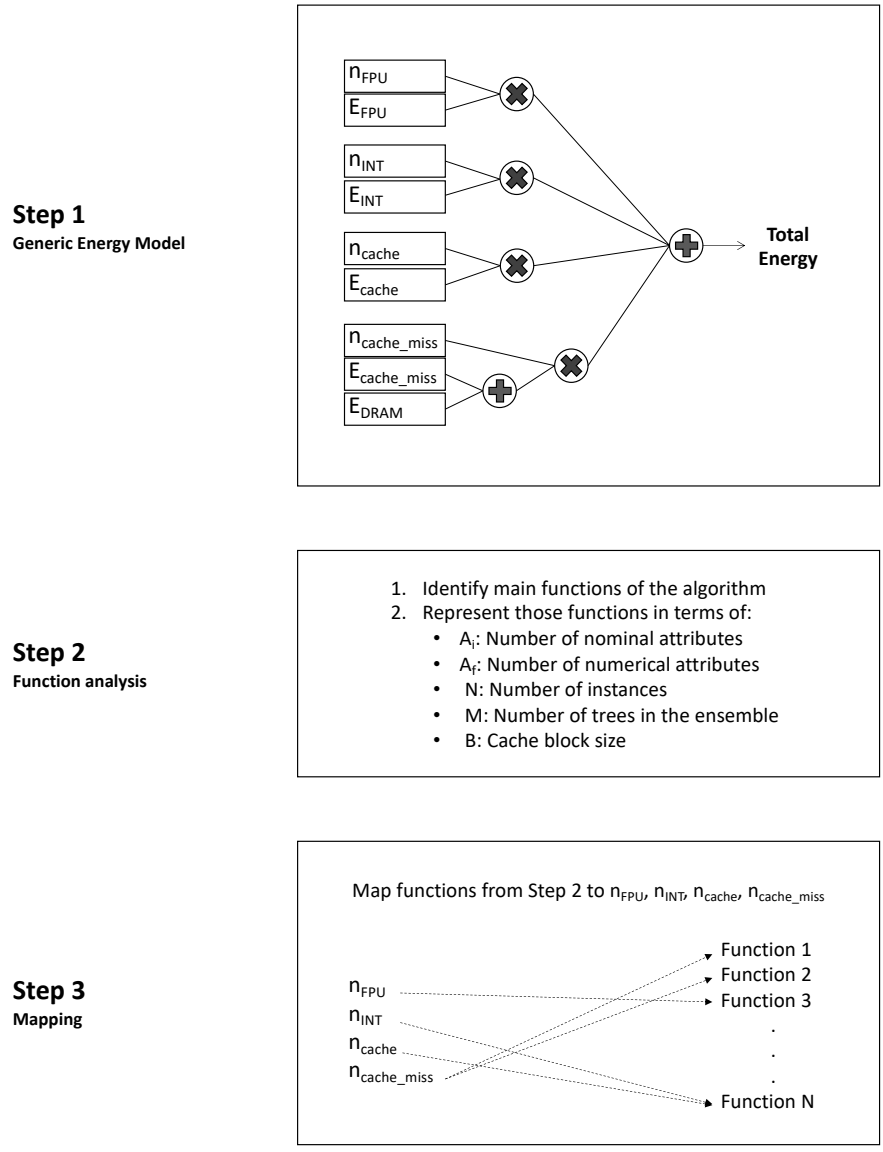

Figure 4.1: *Generic approach to build energy models for any kind of algorithm*

the tree, we know that this function will be executed once per instance, per attribute (considering each node as an attribute). This step is similar to

the time complexity analysis in the form of the big O notation, where we represent an algorithm in terms of the number of iterations. The last step is to map these functions to number of floating point operations, integer operations, number of cache accesses, and number of cache misses. Any cache miss will be calculated as the energy cost of the miss plus the cost of accessing the DRAM. After this step, we have an overview of how energy is spent, by also understanding the differences in energy consumption between a DRAM access and an integer operation [47].

4.2 Energy Efficient Analysis of Hoeffding trees

This section describes in detail how Hoeffding trees consume energy. We first apply the generic energy model from Figure 4.1 to create a theoretical energy model for Hoeffding trees, detailing the main functions of the algorithm and the energy hotspots. We then explain the effect on energy consumption of tuning the different parameters. We finally present a sensitivity analysis of how increasing the number of attributes and instances affect energy consumption.

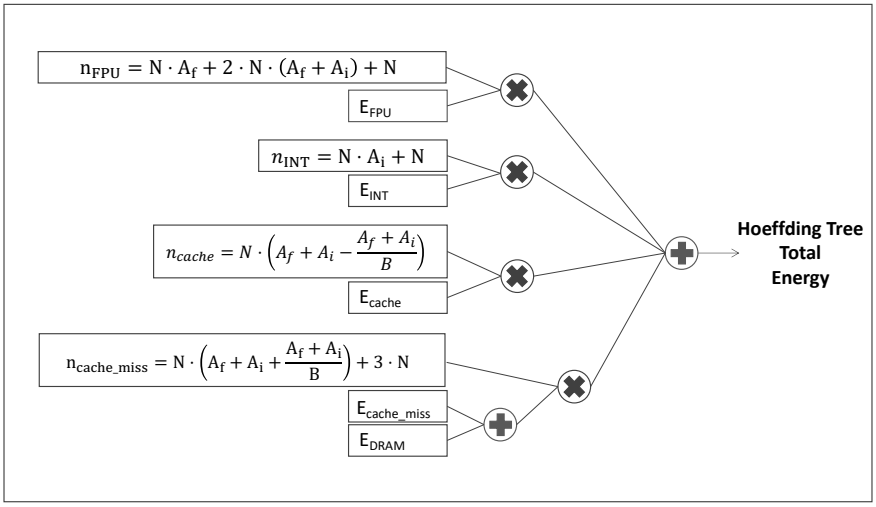

Figure 4.2: *Energy model for Hoeffding Trees based on the steps from Figure 4.1*

Energy model of Hoeffding trees Figure 4.2 shows the energy consumed by Hoeffding trees divided into the different types of computations and

memory accesses. To obtain this model, we identified the main functions of the algorithm (Alg. 1), and mapped them to type of operations, described as follows (explained in detail in Papers IV and V).

The main functions of the algorithm are: traversing the tree, updating the statistics, checking for a split and doing a split. The energy spent on **traversing the tree** is calculated as the number of cache misses, which we assume as one access per attribute per instance (for every time an instance is read), as the worst case scenario. The energy spent on **updating statistics** is divided in nominal and numerical attributes. The energy spent on updating attributes regarding computations is one integer computation per instance per nominal attribute, and one floating point operation per numerical attribute. In terms of accesses, for both nominal and numeric attributes, we have a cache miss the first time we access the table, one cache hit per nominal attribute, and then a cache miss for every attribute that exceeds the block size. To **check for a split** we need to calculate the entropy for all attributes, calculate the Hoeffding bound, and sort the attributes to obtain the best one. Calculating the entropy, calculating the Hoeffding bound, and sorting the attributes, is one floating point operation per attribute, every *nmin* instances. Regarding memory accesses, we consider one cache miss every *nmin* instances. To create a **new node** (do a split) we consider one floating point operation and one cache miss every *nmin* instances.

We combine this operations and memory accesses into the model in Figure 4.2. More step by step details to obtain the final equations are presented in Papers IV and V.

Energy hotspots In Paper III we did an empirical analysis of the most energy consuming functions of Hoeffding trees. We discovered that those functions are related to labeling the leaves of the tree with a specific class, updating the statistics at the leaf, and calculating the best attributes. Updating the statistics and labeling the leaves are more energy consuming because it is has to be done every instance, due to the evaluation method, while calculating the best attributes is done every *nmin* instances. That is the reason why we focus on reducing even more the energy consumed in obtaining the best split attribute. This matches with the results of the energy model, where handling attributes has a significant impact on energy

consumption.

Parameter tuning Varying the parameters of the Hoeffding tree algorithm has an impact on accuracy and energy consumption, studied in Paper II. The results of such paper show that, on average, a higher value of *nmin* results in lower accuracy, and lower energy consumption, since less energy is spent on splits. A higher value of τ, the tie breaking parameter, results in more splits, increasing accuracy, while energy increases or decreases for different datasets. A lower value of δ results in higher accuracy and lower energy consumption. The δ parameter represents the minimum confidence level that is needed to create a split, presented in the Hoeffding bound equation (Chapter 2, Eq. 2.1). Setting the memory limit to a smaller value results in less accurate and less energy consuming trees, as expected. We also test for a different splitting criteria, the Gini index [73], which results in significantly less accuracy and higher energy consumption.

Other interesting results are related to accuracy, size of the tree, and energy consumption. There is no direct correlation between accuracy and energy consumption, since a higher accuracy does not always require a higher energy consumption. This matches with the findings in Paper VI, where we show different ways of growing the tree that result in higher accuracy and less energy consumption compared to its competitor. On average, the bigger the tree the higher the energy consumption. However, with the Green Accelerated Hoeffding Tree, we show that smaller trees can achieve a higher accuracy, by choosing the key nodes that most contribute to accuracy increases.

Sensitivity analysis Paper IV presents a sensitivity analysis of how increasing the number of instances, numerical, and nominal attributes impact accuracy and energy consumption.

Figure 4.3 shows that the increase of energy consumption per instance is non-linear. Increasing the number of instances from 1 million to 10 million increases the energy consumption more than 10X, compared to the low increase from 100,000 to 1 million. One of the reasons is that once the number of instances is high enough, the tree starts growing faster, and the statistics at each node might not fit in cache anymore, having to access the DRAM memory. This results in higher energy consumption.

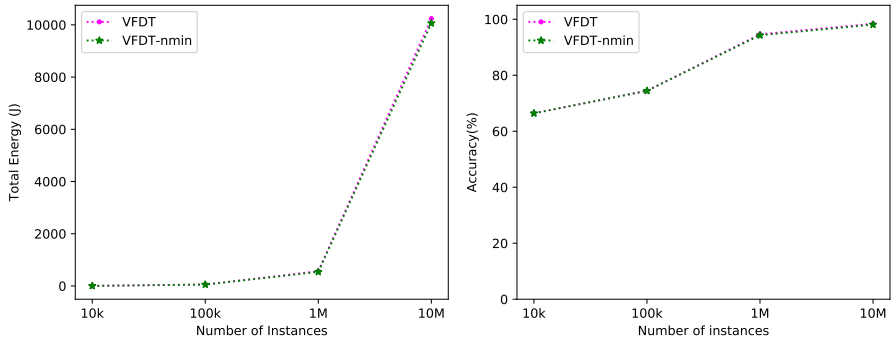

Figure 4.3: *Increase of accuracy and energy consumption per number of instances for the VFDT and the VFDT-nmin*

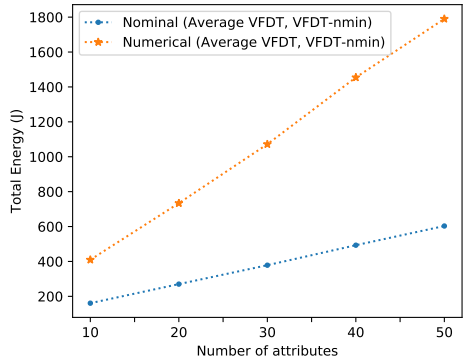

Figure 4.4: *Comparison between numerical and nominal attributes in terms of energy consumption per number of attributes*

Figure 4.4 shows a comparison on energy consumption between nominal and numerical attributes. Numerical attributes consume significantly more energy than nominal attributes, i.e. 600 J for 50 nominal attributes against 1800 J for 50 numerical attributes. That difference increases when the number of attributes increases.

Algorithm 4 VFDT-*nmin*: Very Fast Decision Tree with *nmin adaptation*.
Symbols: HT: Initial tree; X: set of attributes; $G(\cdot)$: split evaluation function; τ: tie threshold; *nmin*: batch size

1: **while** stream is not empty **do**
2: Read instance I_i
3: Sort I_i to corresponding leaf l using HT
4: Update statistics at leaf l
5: Increment n_l: instances seen at l
6: **if** $nmin \leq n_l$ **then**
7: Compute $\overline{G_l}(X_i)$ for each attribute X_i
8: X_a, X_b = attributes with the highest $\overline{G_l}$
9: $\Delta\overline{G} = \overline{G}_l(X_a) - \overline{G}_l(X_b)$
10: Compute ϵ
11: **if** $(\Delta\overline{G} > \epsilon)$ or $(\epsilon < \tau)$ **then**
12: Replace l with a node that splits on X_a
13: **for** each branch of the split **do**
14: New leaf l_m with initialized statistics
15: **end for**
16: **else**
17: Disable attr $\{X_p|(\overline{G}_l(X_p) - \overline{G}_l(X_a)) > \epsilon\}$
18: **if** $\Delta\overline{G} \leq \tau$ **then**
19:
20: $nmin = \left\lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot \tau^2} \right\rceil$
21: **else**
22: $nmin = \left\lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot (\Delta G)^2} \right\rceil$
23: **end if**
24: **end if**
25: **end if**
26: **end while**

4.3 Energy Efficient Splitting Criteria of Hoeffding trees

Very Fast Decision Tree with *nmin adaptation* The Very Fast Decision Tree with *nmin adaptation* is presented in Alg. 4. As was previously explained, the *nmin adaptation* adapts the value of the *nmin* parameter (lines 20 and 22 from Alg. 4) to avoid computing the best attributes when the

information observed is not enough to make a confident split. We apply this method to standard Hoeffding trees (Very Fast Decision Tree algorithm, Paper IV) and to ensembles of Hoeffding trees (Leveraging Bagging, Online Coordinate Boosting, Online Accuracy Updated Ensemble, Online Bagging, and Online Boosting, Paper V).

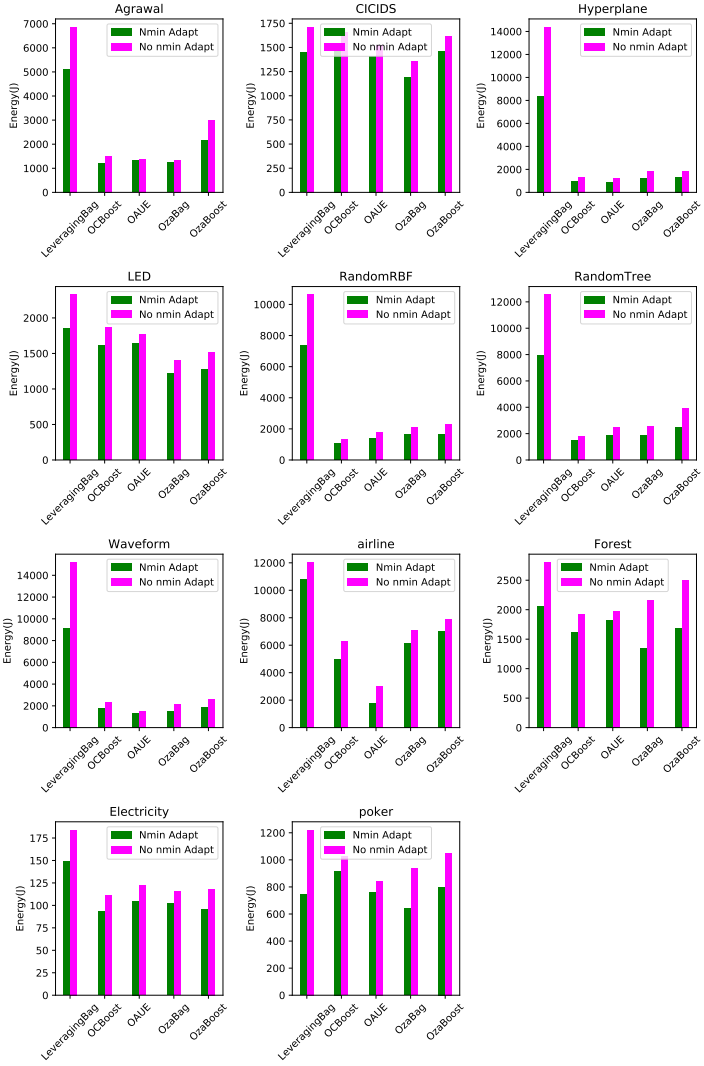

Figure 4.5: *Energy consumption comparison between different ensembles of Hoeffding trees with and without nmin adaptation*

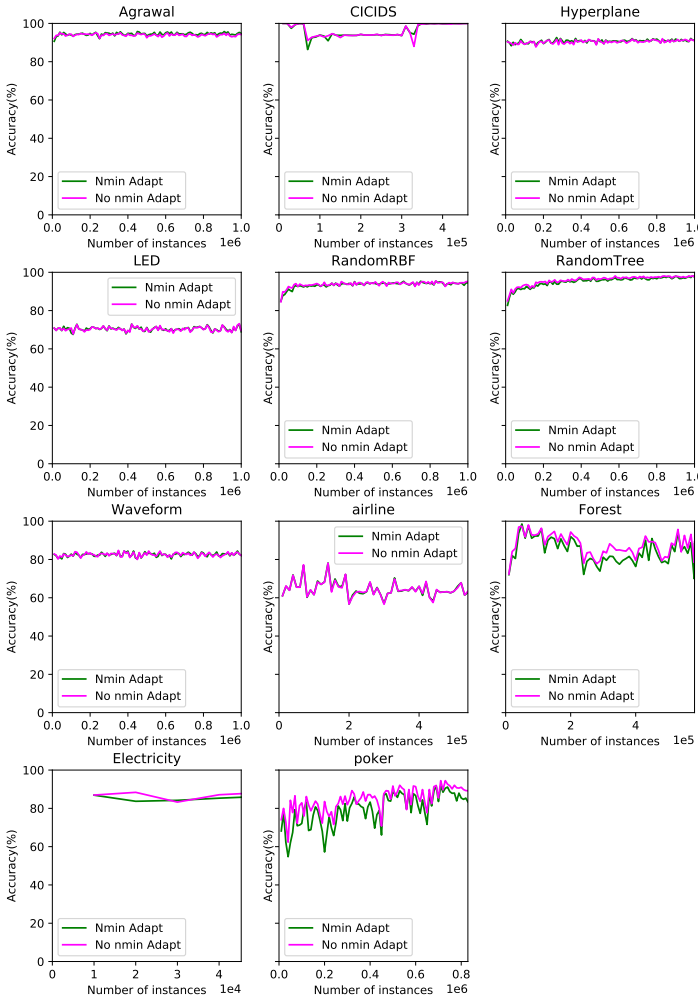

Figure 4.6: *Accuracy comparison averaged for all ensembles of Hoeffding trees with and without nmin adaptation*

The results of Paper IV show that the VFDT-*nmin* consumes significantly less energy than the VFDT, and that the difference between the accuracy results from both algorithms is not statistically significant. Figure 4.5 shows the difference in energy consumption for the ensembles of Hoeffding trees with and without *nmin adaptation*, taken from Paper V. The results of Figure 4.5 show how the the algorithms with *nmin adaptation* consume less energy than the original version with no *nmin adaptation*. Accuracy is

Algorithm 5 Green Accelerated Hoeffding Tree. **Symbols:**HT: Initial tree; X: set of attributes; $G(\cdot)$: split evaluation function; τ: tie threshold; $nmin$: batch size

while stream is not empty **do**
  Read instance
  Traverse the tree using HT
  Update statistics at leaf l
  Increment n_l: instances seen at l
  $fraction = \frac{n_l}{n_{since\_creation}/n_{leaves}}$
  if $fraction <$ deactivateThreshold **then**
    DeactivateLeaf
  else if $fraction >$ growFastThreshold **then**
    growFast $\leftarrow$ True
  end if
  if $nmin \leq n_l$ **then**
    Compute $\overline{G_l}(X_i)$ for each attribute X_i
    if growFast == True **then**
      X_b = null split
    else
      X_b = second best attribute
    end if
    $\Delta\overline{G} = \overline{G}_l(X_a) - \overline{G}_l(X_b)$
    if $(\Delta\overline{G} > \epsilon)$ or $(\epsilon < \tau)$ **then**
      Split $\leftarrow$ True
    end if
    if Split==True **then**
      CreateChildren(l)
    else
      Disable att $\{X_p|(\overline{G}_l(X_p) - \overline{G}_l(X_a)) > \epsilon\}$
    end if
  end if
end while

hardly affected, as can be observed in Figure 4.6.

Green Accelerated Hoeffding Tree The Green Accelerated Hoeffding Tree (GAHT) is presented in Paper VI as an energy-efficient extension to the Extremely Fast Decision Tree (EFDT) [34]. The GAHT (Alg. 5) grows the

tree dynamically, creating three different types of nodes:

- Standard nodes: These nodes grow following the same splitting and growth criteria as standard Hoeffding trees.
- Fast-growing nodes: When the fraction of observed instances at a node is higher than a specific threshold (line 9 in Alg. 5), the standard node becomes a fast-growing node. Fast-growing nodes follow the EFDT splitting criteria, creating less restrictive splits, which results in the tree growing faster on those branches.
- Inactive nodes: When the fraction of observed instances at a node is less than a specific threshold (line 7 in Alg. 5), that node is deactivated. Deactivated nodes still save the current and new information observed at such nodes but are not allowed to grow.

The resulting algorithm grows the tree individually for each branch. The branches that observe more information are allowed to split more easily, contributing to a faster growth in that branch. On the other hand the branches that are observed less are deactivated, pausing the growth and saving energy.

In this paper we show that EFDT creates significantly bigger trees than the VFDT, incurring in higher energy consumption, both on computations and memory space. However, the GAHT is able to build smaller trees while still achieving similar or higher levels of accuracy. This is visible in Figure 4.7.

Figure 4.8 shows the comparison between the VFDT, EFDT, and GAHT in terms of energy efficiency. Energy efficiency, introduced in Chapter 2, is defined as $\frac{energy}{kappa}$, where *kappa* is the difference in accuracy between the algorithm and if the algorithm would choose the majority class as the label. We observe how GAHT is more energy efficient than EFDT on all datasets, and more than the VFDT on half of the datasets.

4.4 Summary

The key results of this thesis are summarized as follows:

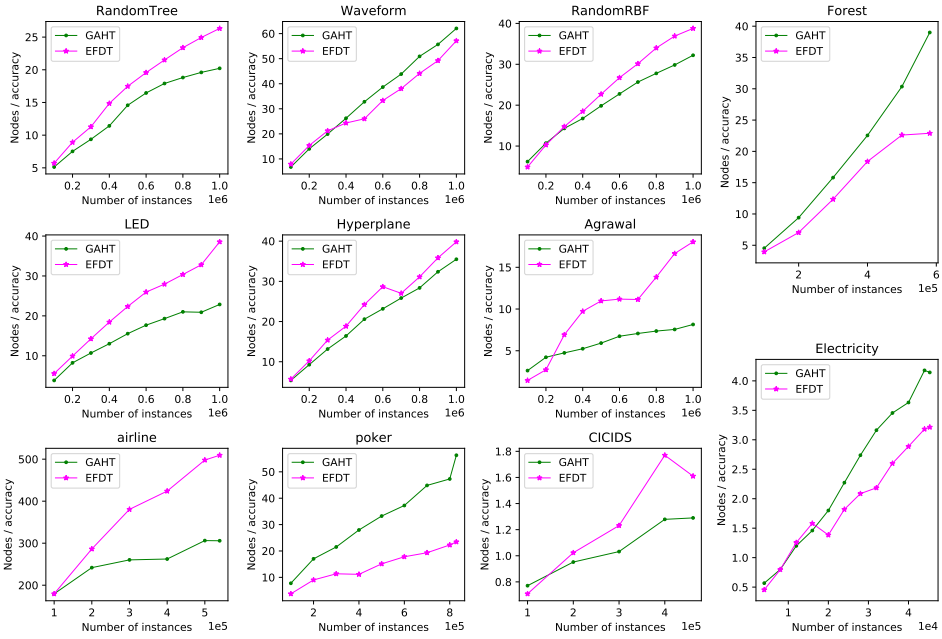

Figure 4.7: *Tree size comparison per number of instances for the GAHT and EFDT algorithms. The nodes/accuracy measurement represents how many nodes are needed to achieve a specific accuracy. A lower value represents a lower number of nodes, which is the desired output.*

- The energy consumption patterns of an algorithm can be obtained both theoretically and empirically. Theoretically through an energy model that maps the number of computations and memory accesses to the main functionalities of the algorithm. Empirically through analyzing the hardware events correlated to the execution of the algorithm, or by analyzing the behavior of the algorithm through functional analysis and parameter tuning.

- The energy consumed by online decision trees, and Hoeffding trees in particular, can be reduced by growing the tree dynamically for each branch. This is done by: i) setting adaptive parameters for each node, so that each branch grows independently from the others by having unique parameter values; and ii) setting independent splitting criteria for each node, depending on the data that such node has observed.

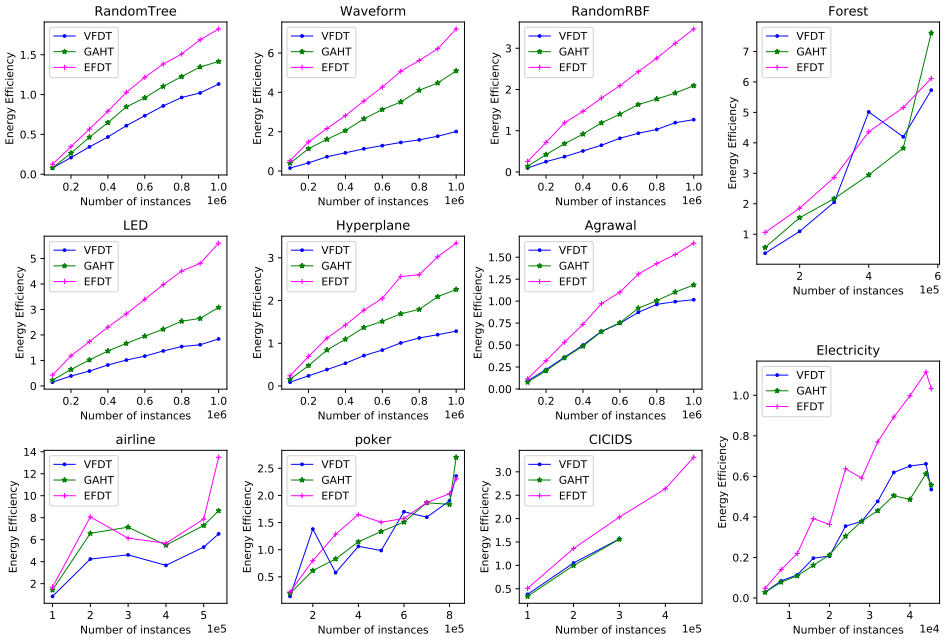

Figure 4.8: *Energy efficiency comparison between the VFDT, GAHT, and EFDT. Energy efficiency is calculated as* $\frac{energy}{kappa}$*, representing how much energy is needed to achieve a specific kappa statistic value.*

5

Conclusions and Future Work

This thesis comprises six research papers, which address *how to design machine learning algorithms that automatically learn from streaming data in an energy-efficient manner.* First, we show how to estimate energy consumption of machine learning algorithms through a survey that maps energy estimation techniques from computer architecture to machine learning scenarios. We use that knowledge to, first, present our own generic approach to build theoretical energy models for any class of algorithms and in particular for Hoeffding trees, and second, to extract the energy consumption patterns for that class of algorithms. Then, we present two approaches to reduce the energy consumption of Hoeffding trees and of ensembles of Hoeffding trees, yielding energy reductions of 30 and 20 percent respectively. These solutions focus on the concept of growing the tree adaptively and independently for each branch. In particular, the first solution uses the *nmin adaptation* method to set a unique value of the *nmin* parameter (minimum batch size observed at each node) for each node, to avoid unnecessary split checks. The second solution is a new algorithm, the Green Accelerated Hoeffding Tree, which grows the tree dynamically by creating different types of nodes which follow different splitting criteria. Thus, growing the tree faster in only a selected set of nodes and deactivating another set of nodes to save energy.

Propositions for future work center around the areas of embedded and IoT devices, where energy is a primary design constraint. The environmental impact of machine learning applications has been studied by several researchers [43]. It is only natural to continue advancing machine learning into a greener future where the energy impact of the different algorithms are evaluated at the same level as predictive accuracy. Solutions such as TensorFlow Lite [74] are first steps to address this challenge. We believe that this thesis provides the necessary tools to inspire and encourage researchers in the field of machine learning into that greener future.

Bibliography

[1] A. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229.

[2] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, et al. “State-of-the-art speech recognition with sequence-to-sequence models”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 4774–4778.

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97.

[4] N. Jaitly, P. Nguyen, A. Senior, and V. Vanhoucke. “Application of pretrained deep neural networks to large vocabulary speech recognition”. In: *Thirteenth Annual Conference of the International Speech Communication Association*. 2012.

[5] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten. “Exploring the limits of weakly supervised pretraining”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 181–196.

[6] P. O. Pinheiro, R. Collobert, and P. Dollár. “Learning to segment object candidates”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 1990–1998.

[7] S. Lamkhede and S. Das. “Challenges in Search on Streaming Services: Netflix Case Study”. In: *In Proceedings of SIGIR ’19: SIGIR Symposium on IR in Practice (SIRIP) 2019 (SIGIR ’19)*. ACM. 2019.

[8] J. Bennett, S. Lanning, et al. “The netflix prize”. In: *Proceedings of KDD cup and workshop*. Vol. 2007. New York, NY, USA. 2007, p. 35.

[9] C. C. Aggarwal. *Data streams: models and algorithms*. Vol. 31. Springer Science & Business Media, 2007.

[10] A. Bifet and R. Kirkby. “Data stream mining a practical approach”. In: (2009).

[11] A. Bifet, R. Gavaldà, G. Holmes, and B. Pfahringer. *Machine Learning for Data Streams with Practical Examples in MOA*. `https://moa.cms.waikato.ac.nz/book/`. MIT Press, 2018.

[12] J. G. Koomey, S. Berard, M. Sanchez, and H. Wong. “Assessing trends in the electrical efficiency of computation over time”. In: *IEEE Annals of the History of Computing* 17 (2009).

[13] P. N. Whatmough, C. Zhou, P. Hansen, S. K. Venkataramanaiah, J.-s. Seo, and M. Mattina. “FixyNN: Efficient Hardware for Mobile Computer Vision via Transfer Learning”. In: *arXiv preprint arXiv:1902.11128* (2019).

[14] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma. “FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 9031–9042.

[15] C. F. Rodrigues, G. Riley, and M. Luján. “SyNERGY: An energy measurement and prediction framework for Convolutional Neural Networks on Jetson TX1”. In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2018, pp. 375–382.

[16] P. Domingos and G. Hulten. “Mining high-speed data streams”. In: *Proc. 6th SIGKDD Int’l Conf. on Knowledge discovery and data mining*. 2000, pp. 71–80.

[17] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn. "Estimation of energy consumption in machine learning". In: *Journal of Parallel and Distributed Computing* 134 (2019), pp. 75–88. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2019.07.007.

[18] E. Garcia-Martin, N. Lavesson, and H. Grahn. "Energy Efficiency Analysis of the Very Fast Decision Tree Algorithm". In: *Trends in Social Network Analysis* (2017), p. 229.

[19] E. Garcia-Martin, N. Lavesson, and H. Grahn. "Identification of energy hotspots: A case study of the very fast decision tree". In: *International Conference on Green, Pervasive, and Cloud Computing*. Springer. 2017, pp. 267–281.

[20] A. M. Turing. "Computing machinery and intelligence". In: *Mind* 59.236 (1950), pp. 433–460.

[21] J. McCarthy, M. Minsky, C. Shannon, N. Rochester, and D. College. "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence". In: (1955).

[22] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[23] T. M. Mitchell. "Machine learning. 1997". In: *Burr Ridge, IL: McGraw Hill* 45.37 (1997), pp. 870–877.

[24] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[25] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization." In: *ICISSP*. 2018, pp. 108–116.

[26] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer. "Efficient online evaluation of big data stream classifiers". In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2015, pp. 59–68.

[27] A. Bifet and R. Gavaldà. "Adaptive learning from evolving data streams". In: *International Symposium on Intelligent Data Analysis*. Springer. 2009, pp. 249–260.

[28] E. S. Page. "Continuous inspection schemes". In: *Biometrika* 41.1/2 (1954), pp. 100–115.

[29] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. "Learning with drift detection". In: *Brazilian Symposium on Artificial Intelligence*. Springer. 2004, pp. 286–295.

[30] E. Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[31] W. Hoeffding. "Probability inequalities for sums of bounded random variables". In: *Journal of the American statistical association* 58.301 (1963), pp. 13–30.

[32] G. Hulten, L. Spencer, and P. Domingos. "Mining time-changing data streams". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 97–106.

[33] A. Bifet and R. Gavalda. "Learning from time-changing data with adaptive windowing". In: *Proceedings of the 2007 SIAM international conference on data mining*. SIAM. 2007, pp. 443–448.

[34] C. Manapragada, G. I. Webb, and M. Salehi. "Extremely fast decision tree". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2018, pp. 1953–1962.

[35] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet. "A survey on ensemble learning for data stream classification". In: *ACM Computing Surveys (CSUR)* 50.2 (2017), p. 23.

[36] Y. Freund and R. E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[37] N. C. Oza and S. Russell. "Experimental comparisons of online and batch versions of bagging and boosting". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 359–364.

[38] N. C. Oza. "Online bagging and boosting". In: *2005 IEEE international conference on systems, man and cybernetics*. Vol. 3. Ieee. 2005, pp. 2340–2345.

[39] A. Bifet, G. Holmes, and B. Pfahringer. "Leveraging bagging for evolving data streams". In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2010, pp. 135–150.

[40] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. "New ensemble methods for evolving data streams". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 139–148.

[41] D. Brzezinski and J. Stefanowski. "Combining block-based and online methods in learning ensembles from concept drifting data streams". In: *Information Sciences* 265 (2014), pp. 50–67.

[42] R. Pelossof, M. Jones, I. Vovsha, and C. Rudin. "Online coordinate boosting". In: *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE. 2009, pp. 1354–1361.

[43] E. Strubell, A. Ganesh, and A. McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *arXiv preprint arXiv:1906.02243* (2019).

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[45] M. Dubois, M. Annavaram, and P. Stenström. *Parallel computer organization and design*. Cambridge University Press, 2012.

[46] N. Weste, D. Harris, and A. Banerjee. "Cmos vlsi design". In: *A circuits and systems perspective* 11 (2005), p. 739.

[47] M. Horowitz. "1.1 computing's energy problem (and what we can do about it)". In: *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*. IEEE. 2014, pp. 10–14.

[48] M. I. Jordan and T. M. Mitchell. "Machine learning: Trends, perspectives, and prospects". In: *Science* 349.6245 (2015), pp. 255–260.

[49] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[50] B. Goel, S. A. McKee, and M. Själander. "Techniques to measure, model, and manage power". In: *Advances in Computers*. Vol. 87. Elsevier, 2012, pp. 7–54.

[51] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, and V. Boeva. "Hoeffding Trees with nmin adaptation". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2018, pp. 70–79.

[52] D. Brzeziński and J. Stefanowski. "Accuracy updated ensemble for data streams with concept drift". In: *International conference on hybrid artificial intelligence systems*. Springer. 2011, pp. 155–163.

[53] D. J. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

[54] S. S. Shapiro and M. B. Wilk. "An analysis of variance test for normality (complete samples)". In: *Biometrika* 52.3/4 (1965), pp. 591–611.

[55] Student. "The probable error of a mean". In: *Biometrika* (1908), pp. 1–25.

[56] F. Wilcoxon. "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83. ISSN: 00994987. URL: http://www.jstor.org/stable/3001968.

[57] A. Bifet, J. Zhang, W. Fan, C. He, J. Zhang, J. Qian, G. Holmes, and B. Pfahringer. "Extremely Fast Decision Tree Mining for Evolving Data Streams". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2017, pp. 1733–1742.

[58] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. "MOA: Massive Online Analysis". In: *Journal of Machine Learning Research* 11 (2010), pp. 1601–1604. URL: http://portal.acm.org/citation.cfm?id=1859903.

[59] R. Agrawal, T. Imieliński, and A. Swami. "Mining association rules between sets of items in large databases". In: *Acm sigmod record*. Vol. 22. 2. ACM. 1993, pp. 207–216.

[60] M. Harries. *Splice-2 comparative evaluation: Electricity pricing*. Tech. rep. 1999.

[61] E. Ikonomovska. *Datasets*. Retrieved from http://kt.ijs.si/elena_ikonomovska/data.html. Online; accessed 1 December 2019. 2013.

[62] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan. "Cost-based modeling for fraud and intrusion detection: Results from the JAM project". In: *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*. Vol. 2. IEEE. 2000, pp. 130–144.

[63] J. A. Blackard and D. J. Dean. “Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables”. In: *Computers and electronics in agriculture* 24.3 (1999), pp. 131–151.

[64] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier. “Powerapi: A software library to monitor the energy consumed at the processlevel”. In: *ERCIM News* (2013).

[65] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout. “An Evaluation of High-Level Mechanistic Core Models”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* (2014). ISSN: 1544-3566. DOI: 10.1145/2629677.

[66] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. “McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures”. In: *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE. 2009, pp. 469–480.

[67] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. “RAPL: Memory power estimation and capping”. In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. Aug. 2010, pp. 189–194. DOI: 10.1145/1840845.1840883.

[68] S. Desrochers, C. Paradis, and V. M. Weaver. “A validation of DRAM RAPL power measurements”. In: *Proceedings of the Second International Symposium on Memory Systems*. ACM. 2016, pp. 455–470.

[69] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. “Measuring energy consumption for short code paths using RAPL”. In: *ACM SIGMETRICS Performance Evaluation Review* 40.3 (2012), pp. 13–17.

[70] W. R. Shadish, T. D. Cook, and D. T. Campbell. *Experimental and quasi-experimental designs for generalized causal inference*. Wadsworth Cengage learning, 2002.

[71] A. Borg. “On Descriptive and Predictive Models for Serial Crime Analysis”. PhD thesis. Blekinge Institute of Technology, 2014.

[72] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks”. In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138.

[73] C. Gini. "Variabilità e mutabilità". In: *Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi* (1912).

[74] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `http://tensorflow.org/`.

Notice

The included papers have already been published or submitted for publication. The included versions of the papers have been formatted to fit the style of the thesis. In addition, minor typographical errors have been corrected

6

Estimation of Energy Consumption in Machine Learning

Eva García-Martín, Crefeda F. Rodrigues, Graham Riley, Håkan Grahn

Abstract

Energy consumption has been widely studied in the computer architecture field for decades. While the adoption of energy as a metric in machine learning is emerging, the majority of research is still primarily focused on obtaining high levels of accuracy without any computational constraint. We believe that one of the reasons for this lack of interest is due to their lack of familiarity with approaches to evaluate energy consumption. To address this challenge, we present a review of the different approaches to estimate energy consumption in general and machine learning applications in particular. Our goal is to provide useful guidelines to the machine learning community giving them the fundamental knowledge to use and build specific energy estimation methods for machine learning algorithms. We also present the latest software tools that gives energy estimation values, together with two use cases that enhance the study of energy consumption in machine learning.

6.1 Introduction

Computer architecture researchers have been investigating energy consumption for decades, especially to be able to deliver state-of-the-art energy efficient processors. Machine learning researchers, on the other hand, have been mainly focused on producing high accurate models without considering energy consumption as an important factor [1]. This is the case for deep learning, where the goal has been to produce deeper and more accurate model without any constraints in terms of computation. These models have

grown in computation (typically in the GigaFlops) and memory requirements (typically in the millions of parameters or weights). These algorithms require high levels of computing power during training as they have to be trained on large amounts of the data while during deployment they may be used multiple times. Some awareness in energy consumption is starting to arise, originating from a few machine learning research groups [2–5] and challenges such as *The Low Power Image Recognition Challenge* (LPIRC) [6]. Thus, we believe that efforts towards estimating energy consumption and developing tools for researchers to advance their research in energy consumption are necessary for a more scalable and sustainable future.

We believe that the reasons why the machine learning community has not shown more interest in energy consumption is because of their lack of familiarity with the current approaches to estimate energy and the lack of power models in existing machine learning frameworks, for example, in Tensorflow [7], Caffe2 [8], PyTorch [9] and others to support energy evaluations. This study addresses this challenge by making the following contributions:

i. We present a literature review of different energy estimation approaches from the computer architecture community (Section 6.4). We synthesize and classify the papers into high-level taxonomy categories (Section 6.3) and modelling techniques to enable a user from the machine learning or computer architecture community to decide which estimation model could be used or built for a given scenario. We also present the advantages and disadvantages for each category.

ii. We present the current state-of-the-art approaches to estimate energy consumption in machine learning (Section 6.6).

iii. We present the currently available software tools and present their characteristics to the user to facilitate building energy consumption models (Section 6.5). We categorize the tools based on the granularity of the energy estimations, software that is supported, precision etc.

iv. Finally, based on the classification of the surveyed papers we present two use cases from the perspective of a machine learning user that wants to estimate the energy consumption of their machine learning model and reveal insights into the importance of studying energy consumption when designing future machine learning systems (Section 6.7).

Our survey covers energy estimation models but not direct energy measurements since the latter is based on tools such as watt-meters [5] or power sensors [10] for which most systems lack the necessary infrastructure and requires investment in time to set up the necessary hardware and software support [11]. Moreover, some efforts to build energy estimation models require real-time power consumption measurements from such devices and is thus included in our survey. While other works have surveyed energy estimation techniques on mobile devices [12, 13], GPUs [14] and HPC systems [15]; we present, to the best of our knowledge, the first survey of system-level energy estimation approaches in the architecture community that could be applied to machine learning scenarios. We provide a useful guideline to the machine learning community on how to use these well-established methods to estimate energy and build models for their algorithms. This survey extends a previously published paper [16]. The scope of this survey focuses on CPU-based and DRAM-based energy estimations, leaving energy estimation on emerging hardware for future works.

6.2 Background

This section explains the main concepts and terminology used throughout the rest of the paper. **Energy**, measured in joules (J), is the total power consumed during an interval of time. Power, i.e., the rate at which energy is consumed, is the sum of static and dynamic power. **Static power**, also known as leakage power, is the power consumed when there is no circuit activity. **Dynamic power** is the power consumed by the circuit, from charging and discharging the capacitance load in the circuit [17]:

$$P_{dynamic} = \alpha \cdot C \cdot V_{dd}^2 \cdot f \tag{6.1}$$

where α is the activity factor, representing the percentage of the circuit that is active. V_{dd} is the voltage, C the capacitance, and f the clock frequency. The power consumption of a system during an application's execution is a determined by the power consumption of the components themselves as well as the how the components are used. Another metric to evaluate energy efficiency that is used by several surveyed papers is the **energy delay product (EDP)**. EDP is calculated as the product of the energy and the delay (execution time). Since energy is the product of power and time, EDP is the product of power and time squared. This measure is used to give more

importance to application runtime, with the goal of making both low energy and fast runtime applications [18].

Finally, performance counters (PMCs) are a set of special-purpose registers in modern processors that count specific event types that are hardware related (e.g. L2 cache misses). Intel has incorporated many PMCs in their modern processors. However, they differentiate their metrics into **core** and **uncore**. Core refers to the components that are in the core, such as the ALU, registers, L1 cache, and L2 cache. Uncore, on the other hand, are those components outside the processor core, such as the DRAM and memory controllers.

6.3 Taxonomy of power estimation models

Power models are built to design better hardware, design better algorithms or design better software to map these algorithms onto hardware. Following the abstraction levels at the hardware-software stack, there already exists a taxonomy of power models ranging from low-level transistors to high-level system description of hardware components, such as processor, cache, bus and others [19]. In this paper we propose a more detailed taxonomy that better aligns with our goal of mapping energy estimation techniques to machine learning applications. We focus on power estimation models that can be built at the system-level to understand the energy consumption at the application level. We propose the following taxonomy of power estimation models at the system-level:

- Software level: The developers of the model at this level are interested in the energy consumption of the application or software implementation and explore optimization techniques that include designing efficient algorithms or better software implementation of the algorithm.
  - Application-level: At the topmost abstraction level, a power consumption model can be built by relating algorithmic properties of the application directly to the power estimation. Here, the developer of the power models extracts characteristics of the application, for example, kernel sizes in a neural network, and relate it to the energy profile of the application.
  - Instruction-level: At the next level, a power consumption model can be built to understand which specific instructions in the

program contribute to the energy consumption. The instruction traces can be extracted using an instruction-set simulator or performance counter profiling, and the cost for each instruction can be added either by known or relative the cost of the instruction or experimental data. This can be applied to estimate the energy consumption of the different functions of machine learning algorithms. This is useful for understanding which parts of the algorithm are consuming most of the energy, to focus the efforts on reducing the energy consumption of such parts [20].

- Hardware-level: The developers of the model at this level are interested in the energy consumption of specific hardware components. They are interested in identifying the hardware components (processor, memory and IO peripherals) that are strongly correlated to the power of the application, also referred to as Functional-level power analysis (FLPA) in [21]. These power models are valuable for the machine learning researchers interested in building specific chips for machine learning computations [22].

There are a number of ways to extract useful features or activity factors from the target hardware. These include simulators and performance counters. More details about these techniques and their connection to the proposed taxonomy categories are given in Section 6.4.

6.4 Approaches to estimate energy consumption

The goal of this section is to introduce key approaches to estimate energy to the machine learning expert. First, we give a general overview of the energy estimation field, providing the reader with the basic knowledge. Second, we explain in detail how the energy estimation models are built, to give the machine learning expert or computer architecture researcher the starting point to build or use more specific machine learning energy models.

The papers reviewed in this section are chosen from a more general survey [11] on power measurement, estimation, and management. We also include papers that we discovered from researching the field on ways to estimate the energy consumption that could be applied in machine learning scenarios. This was achieved by searching on online databases (e.g. Google Scholar) and by looking at the references from some key papers in the area.

The surveyed papers can be clustered into four groups: i) papers that obtain the activity factors with performance counters and use regression or correlation techniques to obtain the power or energy; ii) papers that use simulation data to obtain the activity factors; iii) papers providing architecture or instruction level information; iv) papers that provide real-time power or energy estimation. Table 6.1 provides the connection between the general categories from the taxonomy presented in Section 6.3 and the specific techniques mentioned above (PMC, simulation, architecture or instruction level, and real-time estimation).

Table 6.1: *Connection between the categories from the taxonomy (Section 6.3) and the techniques from Section 6.4*

Taxonomy		Technique
Software-level		
Application-level	→	PMC, Simulation, Real-time power estimation
Instruction-level	→	Instruction-level estimation
Hardware-level	→	Hardware-level estimation

Table 6.2: *Advantages, disadvantages, and possible machine learning applications of the techniques summarized in Sections 6.4.1, 6.4.2, 6.4.3 and 6.4.4*

Technique	ML Application	Advantages	Disadvantages
PMC	Energy consumption analysis of any ML model	No overhead. Application independent	No per-processor results
Simulation	Analysis of algorithms behaviour on ML specific hardware	Detailed results	Significant Overhead
Instruction-level	Energy consumption analysis of specific layers in a neural network	Detailed breakdown of energy consumption	Not easily available
Architecture-level	Improve programming hardware for ad-hoc ML applications	Detailed view	Usually not generalizable to different hardware platforms
Real-time	Streaming data and IoT	Easily available	Usually not detailed results

Table 6.2 summarizes the advantages and disadvantages of each technique, together with examples of possible machine learning applications of such techniques. More details are given in Sections 6.4.1, 6.4.2, 6.4.3 and 6.4.4, where we provide a synthesis of the reviewed papers grouping them in the mentioned categories.

Table 6.3 categorizes the surveyed papers into: taxonomy category, input, technique, output, validation, model requirements, type of machine, and availability. *Input* refers to what was the input in order to create the model. *Model requirements* refers to the type of activity factors required by the

model to output power or energy consumption values. Most papers require either simulation data or data obtained from performance counters.

6.4.1 Performance counters using regression or correlation techniques

The majority of papers reviewed in this study obtain the activity factors of the computations via performance counters (PMCs), to then build the model using regression techniques. The models that are included in this category are: [3, 23–34].

The majority of the approaches [23–29, 31, 33, 34] derive the power consumption by obtaining the power weights associated to each PMC using linear regression or similar techniques, as presented in Equation 6.2 [28].

$$P_{total} = (\sum_{i=1}^{n_{component}} AR_i \cdot w_i) + P_{static}, \tag{6.2}$$

where w_i is the weight associated to component i, AR_i is the activity ratio of component i, and P_{static} represents the overall static power of all components.

While not all papers in this category follow Equation 6.2, their power modelling approach can be approximated by it with a few modifications. In particular, a few approaches [27, 29] present a piecewise linear regression model. They divide the linear function into several segments that better fit the data. To obtain the weights associated to the components, all papers mentioned above except for two [31, 34], isolate the power associated by each component by running a specific micro-benchmark that only runs instructions that will stress that component and not others. To choose the appropriate set of PMCs, several papers [23, 24, 27, 29, 31] correlate the power consumption to the PMCs, choosing the PMCs with the highest correlation to power. Another study [34], in addition to correlating PMCs to real power measurements, also eliminates the PMCs that show high correlation to other PMCs, choosing PMCs that can isolate power values. They also use a clustering approach to create sets of PMCs, to then choose the most representative PMC from each set. They estimate the total power consumption with regression techniques but adding the frequency and voltage parameters to Equation 6.2.

As opposed to previous models, other approaches [30, 32] correlate capacitance (C value in Equation 6.1) to PMCs. Their approach is similar

to the one represented by Equation 6.2, but in this case the authors' goal is to minimize the difference between the predicted capacitance and the real one. By correlating capacitance instead of power, they obtain models that are independent of frequency and voltage (Equation 6.1). This is different to previous approaches, since they had to create a model for each set of frequency-voltage pair. Rather than using benchmarks that isolate each PMC, they run many benchmarks and obtain the best set of weights from all the runs.

The advantages of using performance counters are many, detailed in Table 6.2. There is practically no extra overhead of using this technique. Thus it can be used to measure the energy consumption of both training and inference of machine learning algorithms. Since it is available for different operating systems (OSX, Windows, and Linux) it can be used in many different scenarios and in many different platforms. Another advantage is that it can be used for both large-scale datasets and small datasets, which is very useful with the current advancements on Big Data. Although the energy results are broken down for the CPU and DRAM, the main drawback is that there is no energy breakdown per process.

6.4.2 Simulation

Other models obtain the activity factors via simulation [21, 35–41]. Wattch [35] was one of the first architectural simulators that estimated CPU power consumption. They presented parametrized power models and used analytical dynamic power equations to estimate the power values. Their models are based on capacitance estimations, similar to the papers mentioned above [30, 32]. However, instead of correlating capacitance to performance counters, Wattch estimates the capacitance at the circuit level. Another approach that estimates capacitance using simulation is SimplePower [36], which provides a table that characterizes the capacitance for each *input transition*. The total energy for each module is calculated as the sum of the energy consumed by each *input transition* [36]. Furthermore, another study [37] uses PMCs as activity factors, but utilizes Wattch power models to estimate the power consumption. This paper could be also included in Section 6.4.1, where we define PMC-based models.

An extension [38] to Tiwari et al. [42] correlates simulation data to power using regression based approaches. They propose a pipeline-aware energy

model, in contrast to a traditional approach that does not consider the effect of more than one instruction present in the pipeline. Another simulation study that uses regression [39] create piecewise functions to model the non-linearity of the data, similar to the PMC studies already mentioned [27, 29]. The novelty of their approach is that they propose a methodology that uses a reduced set of simulation runs to build their model, reducing the complexity of the solution.

The authors of McPAT [40] present a modeling approach that takes simulation data as input, and based on analytical power models, output power estimates at each unit. McPAT is similar to Wattch, but McPAT provides more state-of-the-art, detailed and realistic models for multi-core systems. Finally, a recent study [41] simulates the number of memory accesses at each level of the memory hierarchy.

Table 6.3: *Energy estimation models*

Model	Taxonomy category	Input	Technique	Output	Model req.	Type of machine	Availability	Validation
[42]	Instruction	Current	Analytical	Energy	Profiling	CPU	Model	Yes
[35]	Hardware	Analytical	Analytical	Power, Energy, Perf	Simu data	CPU	Theoretical	Yes
[36]	Hardware	Analytical	Transition-sensitive	Energy/Power	Simulation	CPU, f units	Theoretical	-
[37]	Hardware	Real	Heuristics	Power	PMC, sim data	CPU	-	Yes
[38]	Instruction	Simulation	Analytical	Energy	Simulation	VLIW Processor	Model	Yes
[23]	Hardware	Real	Correlations	Power, Temperature	PMC	CPU	Model	Yes
[24]	Hardware	Real	Regression	Power	PMC	Intel PXA255	Model	Yes
[25]	Hardware	Real	Correlation	Power, Perf	PMC	Desktop	Model	-
[39]	Hardware	Analytical	Regression	Power, Perf	Simulation	CPU, f units	Methodology	Yes
[26]	Hardware	Real	Liner regression	Power	PMC	Server	Methodology	
[27]	Hardware	Real	Regression	Power	PMC	Desktop	Model	Yes
[40]	Hardware	Analytical	Analytical, empirical	Power, Time, Area	Sim data	Multicore processor	Open source	Yes
[28]	Hardware	Real	Regression	Power	PMC	Multicore processor	Model	Yes
[29]	Hardware	Real	Regression	Power	PMC	CPU	Methodology	Yes
[43]	Hardware	-	-	Energy, power	PMC	Intel processor	Tool	-
[30]	Application	Real	Correlation	Power, Perf	PMC	Processor	Model	Yes
[44]	Instruction	Real	Analytical	Energy	PMC	Intel Xeon Phi	Methodology	Yes
[21]	Hardware	Real	Regression	Power	Simulation	m-CPUs, mem, f units	Theor, Method.	Power data
[45]	Instruction	Compile time	Regression	Perf, energy, EDP	Static code analysis	mCPU	Methodology	Sandy bridge
[31]	Hardware	Real	Regression	Power	PMC, CPU util	mCPUs (2 systems)	Theoretical	Power data
[32]	Hardware	Real	Correlations	Power	PMC	CPU	Model	Yes
[33]	Instr, HW	PMC	Linear regression	Static, Dyn energy	PMC	Multicore processor	Model	Yes
[34]	Hardware	Real	Regression, Analytical	Power	PMC	mCPU	Tool	Power data

The main advantages of using a simulation approach to estimate energy, as shown in Table 6.2, is that it gives extensive details regarding where the energy is consumed in both the hardware components and at the instruction level. The main drawback is that it introduces a significant overhead, thus big experiments on large-scale datasets are unfeasible. In regards to machine learning, simulation can used to understand which hardware component is responsible for the highest energy consumption, to then design the algorithm accordingly.

6.4.3 Instruction-level or architecture-level estimation

Only a few papers give instruction-level energy estimations [33, 38, 42, 44, 45]. Most of the approaches run a set of curated micro-benchmarks where each benchmark loops over a target instruction type, to be able to isolate the power of that specific instruction [33, 42, 44]. In particular, Tiwari et al. [42] proposes to the best of our knowledge, the first instruction-level energy estimation model. They profile the execution of the program, instead of using performance counters [44]. On the other hand, Shao et al. [44] model the energy per instruction for an Intel Xeon Phi processor, proposing more modern models that consider multi-core and multi-thread processors. The energy is then estimated as a function of the power consumed during the run of the benchmark, the cycle time, and the frequency. Finally, a recent work [33] presents an approach to model energy consumption both at the instruction and architectural levels.

Architectural models are more useful for micro architecture level exploration [44], for instance, to create more energy efficient accelerators for machine learning tasks. The majority of the papers give details at the architectural-level [21, 26, 28, 31, 33–37, 39, 40, 45]. These papers provide an energy or power breakdown per component, thus giving information about which components are the most energy consuming (e.g. L1 cache, DRAM, etc.) These models are also useful for researchers interested in overall power consumption.

6.4.4 Real-time power estimation

All models that obtain the activity factors via performance counters allow for real time energy monitoring. The reason is that accessing those registers does not introduce any significant overhead [3, 23–34, 37, 43, 44]. Some

models, however, need an offline calibration phase to obtain the parameters of the model, but this is usually done only once [26] for each machine.

Simulation based models, on the other side, do not offer real-time energy or power estimation, due to the introduced overhead and that they need to do a full profile run to get the values [21, 35, 36, 38–42].

Real-time estimation is useful for areas such as data stream mining and online learning, where the models are built as the data arrives.

6.4.5 Detailed explanation of the reviewed papers

The previous sections synthesized the surveyed papers byexplaining in general the different categories and main techniques used to obtain power estimation values. A more detailed view into each paper is presented in the following paragraphs. They center on how the model was created, and how can energy be estimated from a few processor statistics.

Tiwari et al.(1996) [42] To our knowledge, this is the first approach to estimate power consumption at the instruction level by assigning an energy cost to each instruction type. Their motivation is to provide an approach to estimate the power consumption of software, which was not done yet at that time, since traditional power analysis tools were not suited for power analysis of software. The total power is given by the current (I) times the supply voltage (V_{DD}). They measure the current directly at the CPU power pins with a meter. Once the current is known, they derive power costs to individual instructions by running specific programs and measuring the current drawn during the execution. They execute a loop that consists of several instances of the same instruction and measure the current and the V_{DD}, obtaining the base power cost for that type of instruction. That base power is multiplied by the number of non-overlapped cycles needed to execute the instruction, obtaining the energy base cost. To the base cost, they add the circuit overhead and the other inter instruction effects such as cache misses. They have applied this methodology on three different commercial processors. They have validated their approach by comparing them with other current measurement setups.

Brooks et al.(2000) [35] Wattch is an architectural simulator that estimates CPU power consumption. They have created power models for

different hardware structures, and then based on the cycle-level simulation data, outputs power consumption, performance (measured as the number of cycles), energy consumption, and energy-delay-product. They have integrated their power models into SimpleScalar, and extended it to obtain more power-related measurements. They validate their models with three approaches. First, they check if they modelled the capacitance levels correctly, by measuring them in real circuits. Second, they compare their power levels with already published results from industry chips. Third, they compare the maximum power of their models against published works.

One interesting aspect of the paper is that they present three case studies where studying power consumption can be useful. In the first case study they vary some architectural parameters (e.g. cache size) and run several benchmarks to compare the power consumption, performance, and energy-delay product. The second case study evaluates the effect of *loop unrolling* on power consumption. The results show how loop unrolling benefits in terms of power and also in terms of energy-delay product, even though after rolling with a factor of 4, the execution time remains the same. This is a clear example where execution time can be the same, however the energy consumption is significantly different. The third case study looks at the effects of *result memoing*, where the inputs and outputs of long-latency operations are stored and re-used the if the same inputs are encountered again. The results show that there is an average power improvement of 5.4%.

Ye et al.(2000) [36] The authors present SimplePower, a simulator that estimates the power and energy consumption per cycle. At each clock cycle, SimplePower simulates the execution of the instructions and outputs power consumption values based on their power models and the usage of the functional units. Their power models are based on input transitions [46]. They simulate a subset of instructions from SimpleScalar and create the energy models and switch capacitance values for the datapath, memory, and on-chip buses. In order to estimate the power/energy consumption of an application, SimplePower simulates the execution of the set of instructions, using the power models associated to each functional unit that have been activated by the set of instructions.

SimplePower provides switch capacitance tables for the following units: ALU, adders, multipliers, shifter, controllers, register file, pipeline registers,

and multiplexors. It also provides a cache simulator based on a modified version of an existing cache simulator [47] and analytical energy models [48]. Finally, they provide a bus simulator, which combines the activity factors from the simulator and an interconnect power model [49] to create the switch capacitance of the on-chip buses.

Joseph et al.(2001) [37] The authors present the Castle project, that gives runtime power readings from different processor units. To the best of our knowledge, they provide one of the first approaches to use performance counters to estimate power consumption. The present per-unit power breakdowns based on direct measurements, rather than through simulations. They partially validate their model with measurements from a power meter. The power model is created using Wattch [35] together with SimpleScalar [50]. They correlate PMCs with the most power-relevant counts. However, the authors do not provide which PMCs are the most relevant ones.

Sami et al.(2002) [38] The authors present an energy model for VLIW (Very Long Instruction word) architectures. *A VLIW processor is a pipelined CPU that can execute, in each clock cycle, a set of explicitly parallel operations [38]* Their model estimates the energy consumption for each instruction, but decomposing it for different active components. It is a pipeline-aware model, since it models the power for each clock cycle, splitting the energy of each instruction into the energy of the pipeline stages. They derive an analytical model to estimate the energy consumption, but we simplify the explanation of the model focusing on the main building blocks. For a complete detailed explanation of the model we ask the readers to refer to the original reference [38].

The main idea is to estimate the total energy consumption by summing the energy contributions of each instruction. The energy consumption associated with a specific instruction is the sum of the contributions of each pipeline stage. That is, the average energy consumed per stage when executing the instruction, plus the energy consumed by the connections between the pipeline stages.

The energy consumed per stage is estimated as a linear combination of the average energy consumption of such state during an ideal execution in the absence of any exceptions, plus the energy consumed at that stage due to a miss on the data cache, plus the additive energy consumed at

that stage due to a miss in the instruction cache. The energy consumed by the connections depends on V_{dd}, the clock frequency, the capacitance, and the switching activity. They are able to estimate the parameters of each equation using linear regression and measuring the power consumption. To validate their model, they obtain real measurements from the Synopsys Design Power tool and compare it to their model by varying the different set of parameters.

Bellosa et al.(2003) [23] The authors provide a model to estimate power consumption and temperature on-the-fly using performance counters. To get the real power measurements they instrument the motherboard with four thermal resistors attached between the board and the power supply. They correlate the real power consumption to processor-internal events to obtain energy estimations. To know which events account for what fraction of the energy consumption, they have a set of test programs (benchmarks) with specific type of operations, such as ALU and memory operations. Running the set of benchmarks will activate a set of PMCs, and will give power consumption values. From the executions of running all the benchmarks, they estimate the weight of each PMC by using linear regression. The output is an energy/power model based on PMC information. The reason why they have this set of benchmarks is clear, to have different activations of different PMCs. If they had very similar programs executions, it would not be possible to correlate the PMC information to energy consumption, and it would not generalize to other type of applications. In this manner, they isolate the power consumption of different type of operations, obtaining more generalizable results. They validate their model with another set of benchmarks and real power consumption measurements.

Gilberto et al.(2005) [24] The authors present a power estimation model that can acquire fast, low-overhead power estimates. They use PMC information to estimation the power consumption of the CPU and memory. To create the model, they correlate real power consumption to five PMCs, using different benchmarks. The goal is to estimate the power weights, that represents the importance of that PMC in relation to the final energy consumption. They also incorporate idle processor power consumption in the power model. To measure the power consumption of the main memory, their initial approach was to use PMCs that monitor memory accesses. However,

the processor of their study did not include that information. Thus, they use PMCs related to instruction cache misses and data dependencies to estimate the power consumption of the main memory. They obtain real power measurements from the processor and memory.

Rajamani et al.(2006) [25] The authors present an application-aware power management methodology. Their methodology incorporates real-time power and performance models for several DVFS frequency-voltage pairs. This is one of the first works that estimates power for different frequency-voltage pairs, since most of the work that we have already presented either focuses only on one frequency level or creates a model for each frequency. The methodology for application-aware power management consists of three phases: i) power and performance monitoring with PMCs; ii) estimation and prediction of power consumption and performance iii) choosing the most optimal voltage-frequency pair. To create the power model for each frequency-voltage pair, they used four micro-benchmarks and one PMC. They believe that the most important counter is *Decoded Instructions per Cycle*, since it correlates highly to measured power. The power estimation model is constructed as a linear fit of measured DPC, minimizing the absolute-value error between the measured power and estimated power.

For the performance model they use the Instructions retired counter, and, based on a set of benchmarks, they create two sets of equations to differentiate between core-bound and memory-bound workloads. To obtain real power measurements to create the models, they have used some meters using a Radisys system board, specified in another study by the same authors [51]. Based on the monitoring and prediction of power and performance, they also present two power management solutions: Performance Maximizer, which sets the most optimal frequency-voltage pair to maximize performance while staying within set power consumption limits. The second solution is Powersave, that provides energy savings while staying within performance set limits. The paper does not give thorough detail on the validation of their models.

Lee et al.(2006) [39] The authors present a model to estimate power and performance by using regression analysis applied on an initial set of simulation of different micro-architectural components specifications. They first choose a set of 4,000 samples of possible specifications, such as L2 cache

size and number of special purpose registers, uniformly at random. They choose a subset of the 22 benchmarks to run their simulations. To derive performance and power models they perform regression analysis on the 4,000 samples.

The idea is to check, from the set of predictors, which are most valuable to predict performance and power. Predictors that are observed to vary similarly via clustering are merged together. Predictors that do not correlate well with performance or power are removed. Since they use regression, in the end the model consists of a set of weights for each predictor, summed together, to obtain the power or performance, depending on the model. The analysis made on the power model state that predictors that are significant for performance prediction are probably also significant for power prediction. They validate their models with the power and performance estimates from the simulator, which also validated their estimates with existing power models [52, 53].

Economou et al.(2006) [26] The authors present Mantis, a method to model the power consumption of server systems using PMCs. Their approach consists of two well defined phases, namely, the calibration phase, and the power estimation phase. During the calibration phase, they correlate real AC power consumption values to system performance metrics (PMCs), using diverse benchmarks as input and linear regression. The calibration phase is done offline and is run only once for each system. Based on the information obtained from the calibration phase, the model estimates power consumption by using real-time PMC information. The power consumption is broken down into the following components: CPU, Memory, Hard disk, Network and peripherals. They are able to obtain the power measurements of these units separately by measuring the real power consumption of the different components of the board during the calibration phase. Finally, to validate their model, they compare the results obtained from Mantis against real AC power consumption in more than 30 benchmarks.

Singh et al.(2009) [27] The authors present a model to estimate power consumption by correlating PMCs to observed power consumption. Based on platform constraints, they restrict the number of studied performance counters to four, namely: *L2_cache_miss*, *retired_uops*, *retired MMX and DF instructions*, *dispatch stalls*. They have chosen these four performance

counters as the ones with highest correlations to real measured power. They form the power model by assigning weights to PMCs. They do this by running micro-benchmarks, and applying multiple linear regression to real power measured with the Watts Up Pro power meter.

They have validated their model by testing it and comparing it to real power measurements on three different benchmarks. The main difference between this work and others, is that they only use the performance counters that are available at run-time in a single run, allowing for real-time power estimation. In comparison to Goel et al. (2010) [29], Goel et.al (2010) extends this work by including temperature in the model, exploiting DVFS, and validating the model on several more platforms.

Li et al. (2009) [40] The authors present McPAT, a power, area, and timing modeling framework. Dynamic power is calculated with analytical models to calculate the capacitance, and statistics from simulation to calculate the activity factors (of how the circuit is being used). They use existing [54] analytical models to calculate the power dissipated from switching the circuits. They use MASTAR [55] and Intel's data [56] to model the leakage power. The model outputs also timing and area results, based on improved versions of existing models. One key improvement in comparison to previous models is that they are able to model power-saving techniques. Their model works such that, based on an input from a simulator, in XML form, which provides activity factors of the different hardware components, outputs power, time, and area values. They validate their results on three different processors, and use the published data to compare the values. Power is validated based on peak power data from the processors that was already published.

Bertran et al.(2010) [28] The authors present an approach to model power consumption using PMCs. Instead of portraying the overall power consumption of the processor, they break down the power distribution per component (floating point unit, integer unit, branch prediction unit, L1 cache, L2 cache, font side bus, and main memory). More specifically, they present a systematic methodology to produce power models for multi-core architectures, focusing on decomposability (per component), accuracy, and responsiveness of the models. Responsiveness they refer to the ability of the model to capture power variations.

Their methodology consists of four steps. First, they define the power components; second, they define the micro-benchmarks; third, they collect the data to train and validate the model; and fourth, they output the model. The power components are a set of micro-architectural components that are grouped together based on levels of activity. For instance, the whole memory subsystem is divided into three power components: L1 cache, L2 cache, and the Front Side Bus (FSB). They match these power components to PMC activity. They design a total of 97 benchmarks with different set of instructions. With the execution of the benchmarks they are able to get the activity ratios of each power component. With power measurements and the activity ratios for each component, they are able to create the power model using multiple linear regression. They validate their models using empirical measurements of the SPECcpu2006 benchmarks [57]. However, it is unclear how they obtain the real measurements to validate or create their models.

Goel et al.(2010) [29] The authors present a methodology to produce per-core power models in real time using PMCs and temperature sensor readings. As the previous models presented above, they first find the PMCs that correlate strongly with measured power. They sample those PMCs while running micro-benchmarks and then apply linear regression to give a weight to each PMC. The best set of PMCs is unique for each system. Their methodology has the following steps. First, they identify the PMCs that represent the heavily used parts of a core's micro-architecture. They derive the following four categories: Floating point units, Memory, Stalls, and Instructions Retired. The instructions retired refer to the number of instructions that were completely executed by the CPU. [1] The *Stalls* category measures resource stalls to understand how the out-of-order logic contributes to power usage, since if an instruction stalls, that can contribute to an increase of dynamic power. Second, they rank the PMCs by running a set of micro-benchmarks that cover the categories mentioned in the first step. They use the Watts Up Pro power meter [58] to obtain real power measurements. They rank the PMCs based on the correlation to the real power measured using Spearman's rank correlation [59]. They used *pfmon* [60] to collect the data for each PMC. Finally, using multiple linear regression they create the power model to estimate the power for each core, introducing also the core

[1] `https://software.intel.com/en-us/vtune-amplifier-help-instructions-retired-event`

temperature. To validate their model they run four benchmark suites, real power measurements with Watts Up Pro power meter, and sensors to obtain core temperatures.

Intel RAPL(2010,2011) [43, 61] Intel's RAPL interface is presented in Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B, Part 2, Chapter 14.9 [61]. It is also referenced in a paper about power estimation in memory [43] using RAPL.

RAPL(Running Average Power Limit) is a driver that allows energy and power consumption readings of the core, uncore, and DRAM. RAPL provides a set of specific PMCs with the energy and power readings. However, the RAPL interface is not well documented since the authors have not published how they model power or energy. There have been several studies that validate RAPL's interface [62, 63], and several tools available that make use of such interface to provide energy measurements, such as PAPI (Performance API) [64]. Since it gives accurate results, and it is available on any Intel modern machine, we have used this interface in our use case in Section 6.7.1.

Spiliopoulos et al.(2012) [30] The authors present a power and performance estimation tool that is per-phase and per-frequency. Their methodology accounts for the different phases of a program, thus being able to understand in more detail which part of the program is responsible for which amount of energy consumption. They provide a performance model based on analytical DVFS models, and a power model based on capacitance correlations. In particular, they collect information from a specific run of an application for each interval/phase using PMCs. To estimate the performance, they use analytical models that are based on the number of stalls and misses for different frequency values. The total power is estimated as the sum of static and dynamic power. The dynamic power is calculated as the frequency (f) times the supply voltage (V_{dd}) squared times the effective capacitance. The effective capacitance is the capacitance times the activity factor. Since they want their model to be applicable to different frequency values, they do not correlate PMC information to power directly (otherwise they would need a model for each frequency), instead, they correlate the core's effective capacitance (which does not depend on frequency) to PMCs. They do this by first running a set of benchmarks in maximum frequency and measuring processor total power consumption, then subtracting the

static power, and finally dividing with $f \cdot V^2$. The static power is measured for all frequencies when the processor is idle. They validate their models using real power measurements.

Shao et al.(2013) [44] The authors present an instruction-level energy model for the Xeon Phi processor using PMCs and analytical models. The total energy is estimated as the energy per instruction (EPI) multiplied by the instruction counts obtained from the PMCs. They collect the PMC statistics using Intel's VTune tool[2]. To calculate the EPI, they use a set of micro-benchmarks that cover all major instruction types. In particular, for each instruction type, the EPI is calculated as the difference in power between the start and the end of the execution of the specific micro-benchmark, times the number of cycles executed by the micro-benchmark, divided by the frequency; and all that term divided by the number of instructions in the micro-benchmark. The power values are real power measurements using the Xeon Phi Beta Software Development Platform. They characterize the EPIs within different number of cores and threads. They show very interesting patterns, such as that power is double for the micro-benchmark that loads a cache line from memory into the local cache, compared to the micro-benchmark that executes an arithmetic operation incurring in no cache misses. Finally, they validate their model using the SHOC benchmark suite [65] and real measurements from their Xeoh Phi card, with 60 cores and four threads.

Rethinagiri et al.(2014) [21] The authors propose to build a power estimation tool at the system-level for embedded platforms. They combine the power models for different components, such as, processor, memory and functional units to obtain a system-level estimate of power consumed.

They first characterize the system by extracting a set of generic features or activities that can be applicable to most programs. These include architectural features such as frequency of the processor, bus, and number of cores, and application features such as instructions per cycle (IPC), cache miss rate and external memory access rate. These performance counter and power information was collected for real systems by running some assembly-level programs to simulate each component. Power values were collected using Agilent LXI digitalizer to obtain static and dynamic power consumption.

[2] `https://software.intel.com/en-us/vtune`

Subsequently, a regression-based approach was use to map these features to the power values. Unlike the previous approaches, they focus on building power models for mobile CPU. Second, they develop a cycle-accurate simulator for different ARM cores (ARM Cortex A9, Cortex-A8 and ARM9) using the component models provided in gem5 simulator. They built single and dual core power model for the Cortex-A9. They extract activities such as instruction miss rates, read and write miss rates and IPC for a benchmark application, that is a JPEG decoder application. Power estimates are provided at a fine-grained level for the main tasks in the decoder application. They validate their power estimation approach to state-of-the art tools such as McPAT with a Mult2Sim functional simulator and real power measurements.

Laurenzano et al.(2014) [45] The authors aim to characterize HPC application benchmarks on several ARM processors in terms of performance, energy, and energy-delay product (EDP). To measure the performance of the HPC kernels in the benchmarks they insert timing instrumentation around main phases of the code such as loops. They measure wall power and attribute this system power to the individual cores analytically. They gather data by running multiple experiments with different core counts and extract the system power based on the number of idle and active cores. They use the empirically gathered data to form a system of equations that can be solved by Gaussian elimination to get the power draw for a single core.

To extract features from the application code they use static binary analysis tools such as EPAX toolkit for the ARM binaries and PEBIL toolkit for x86 binaries to examine the binary and extract information about the machine-level instructions in the program and their relationship with high-level structures in the code, for example, loops. They chose floating point operations, memory accesses number of bytes moved per memory operation, and size of data structures in loops as key features to build a multi-variate regression model for energy consumption. Similar to previous the energy models, they select features that perform well across all benchmarks applications. Hence, these models are useful when comparing different systems at an architectural level to understand changes in the architecture that lead to better energy-use across a range of applications.

Walker et al.(2015) [31] The authors propose building a power model to create an intelligent run-time management software that can use the

information from the power model to apply energy-saving techniques for the applications executing on a mobile CPU. They explore two types of models, one based on PMCs and one based on CPU utilization as features to the model. The main characteristics to build a power estimation model for a run-time management system was chosen on the properties of light-weight, accuracy and responsiveness. They measure the power of the individual CPU as opposed to the whole board using the Agilent N6705 Power analyser. They also measure voltage and current at 10ms intervals. The PMC information and power data was collected for a variety of benchmark applications including MiBench, video playback, and other workloads to exercise certain architectural features.

Since only four PMCs can be monitored simultaneously for the ARM Cortex-A8 on the BeagleBoardxM, multiple experiments were run to capture every PMC. The PMC data was then correlated with power individually to select candidate features in the final power estimation model. The first feature was the one with the highest correlation factor and was used for building the base model. Other PMCs were then selected with the base model as reference. The main goal was to make the PMC-based model applicable to a variety of workloads hence features were selected based on how well it improved the accuracy across all workloads. The final estimation model was then built with the candidate features across multiple DVFS settings. Since one of the characteristics of the model was responsiveness, the power model was built to predict instantaneous power as opposed to average power of the application. Since PMC information is difficult to obtain on embedded platforms, the authors explore the use of CPU-utilization as a candidate feature for the power model. The power model was built for two types of mobile CPU cores (ARM Cortex A7 and Cortex A15) on an Odroid-XU+E board. The board is equipped with voltage and current sensors that were sampled at 50ms. They use regression analysis to correlate the average CPU utilization with the power at multiple DVFS settings. The use the MiBench embedded benchmark suite to exercise both processors. The also experiment with varying number of cores.

Goel et al.(2016) [32] The authors present a methodology to derive static and dynamic power values of individual cores and uncore components. In contrast to previous approaches [29], they use analytical equations together with empirical observations from curated micro-benchmarks to estimate the

effective capacitance. Correlating capacitance with PMCs was already done in a previous study from 2012 [30], as we mention in Section 6.4.1.

They present separate models for the core, uncore, dynamic and static power. They validate their methodology with a set of benchmarks. The methodology uses core and uncore voltage, package temperature, and PMCs to create models that can estimate power consumption for sequential and parallel applications across all system frequencies. Finally, the authors also perform sensitivity analysis of energy consumption at several voltage and frequency levels. They study how energy efficiency relates to DVFS and to the memory intensity. They conclude that memory bound must be considered when choosing an optimal frequency to obtain the most energy efficient setup.

Mazouz et al.(2017) [33] The authors present a methodology to derive energy models of the hardware components of multi-core processors. They calculate the total energy of the CPU as the sum of the static and dynamic energy. The static energy is calculated by first calculating the static power of the core (ALU, FPU, L1 cache, L2 cache, etc.) and uncore (L3 cache) components. The dynamic energy is calculated as the sum of the energy estimated for each hardware component. The hardware components are the following: FE (front-end, number of instructions issued to the back-end), INT (integer instructions related energy consumption), LD (memory loads L1 cache), ST (memory store L1 cache), FP (floating point instructions related energy consumption), L2 cache, L3 cache. To estimate the energy of each component, the authors run a set of micro-benchmarks to isolate the power of each component. They apply linear regression to obtain the weights for every component. The energy measurements are obtained using PMC information and the interface RAPL. They claim that the methodology is flexible to use any other source of power measurement as input. The interesting part of their model is that they not only provide accurate energy values per-component, but also for each instruction type, giving a nice overview of the energy cost per specific type of operation. They do this by running a set of micro-benchmarks with specific instructions and isolating their power and energy values. This allows for code optimization in terms of energy efficiency. They validate their models by using a different set of benchmarks that run different set of operations.

Walker et al.(2017) [34] The authors build a run-time power model for ARM's Cortex A7 and A15 for the purpose of run-time power management and design space exploration for the application. They perform a thorough investigation of PMC event selection using statistical techniques such as hierarchical clustering and R^2 analysis to capture the relationship between power and the PMC data. They also measure collinearity of the PMCs with each other to avoid duplicating information that is used for building the power models.

They use the optimal PMC events from the PMC event selection phase and use correlation techniques to relate these events to the clock frequency and CPU Voltage. They model the power at the cluster-level which includes multiple cores and their caches. They also built models for varying number of cores and different core-affinities. Their benchmarks include 60 workloads curated from different sources such as MiBench, LMBench and others, to exercise the CPU, memory and the I/O.

6.5 Power and performance monitoring tools

This section provides an overview of available tools that can facilitate building power models. The tools are characterized on the following criteria:

- Language interface of the tool: How can the tool be accessed and used? Has the developer of the tool provided a set of Command-Line Interface (CLI) options or a Graphical User Interface (GUI)?
- Research Paper: The research paper were the tool is presented.
- Operating systems supported: Is there support for different types of operating systems. MacOS (M), Linux (L) and Windows (W).
- User-friendly: This will depend on the language interface provided and the availability of documentation of the tool.
- Maturity: The duration since the establishment of the tool and the technical support for the tool.
- Research or commercial: Is the tool the result of a research endeavour or a commercial product?

ARM Streamline Performance Analyser can be used to monitor the power profile and performance counter for mobile CPUs based on the ARM architecture. The tool provides both graphical and command line interfaces to obtain real power values on the target device but has to be interfaced with the necessary power measuring equipment such as an ARM energy probe or the power sensors on-board. Moreover, the tool does not provide energy estimation models based on the data it collects. A recent effort in this direction was made by SyNERGY [3] that leveraged the tool for building energy estimation models[3]. Finally, the reported overhead of using ARM Streamline with gator daemon (the latter runs on the target device) is within 0.5 to 3% error.

Intel Power Gadget uses the Intel RAPL interface [61] to provide power and energy estimations of the core and uncore of the processor, together with the DRAM. It has both a GUI and a command line tool, and they provide an API to extract information from sections of code. The API is limited to C/C++ code. The command line tool can be used to obtain real time energy values during the execution of a specific script or command. The script can contain runs of any programming language. However, the energy values correspond to the energy of the whole processor, not only of the energy responsible for that particular application. Thus, our recommendation is to make energy estimations while no other application is running in the background, and comparing several executions under the same setups. Reported errors claim that RAPL gives results within 2.3% of actual measurements for the DRAM; and that RAPL slightly underestimates the power for some workloads [62].

McPAT can be used together with Sniper [66] to simulate the execution of a C application. McPAT outputs power and energy consumption values separately for the different components: FP unit, L2 cache, etc. C code can be instrumented to obtain power measurements at the functional level. McPAT gives more fine granular information compared to Intel Power Gadget, giving energy estimations at the application, hardware, and functional level. However, only tasks that require a low number of instructions can be executed, since it introduces a significant overhead. For instance, machine learning algorithmic runs can be executed as long as the datasets are small (≈100k instances, 50 attributes). McPAT reported errors range between 10 and 20

[3] An initial version of the framework is available here: `https://github.com/Crefeda/SyNERGY`

percent depending on the processor.

Table 6.4: *Description of energy measurement and estimation tools*

Tool	Research Paper	Language interface	Operating systems	User-friendly	Maturity	Research / commercial	Uses
ARM Streamline	[10]	CLI and GUI	W,L	Yes	2010	Commercial	Power, PMU
Powmon	[34]	CLI	M,W,L	Not tested	2017	Research	Power, PMU
Intel Power Gadget	[61]	CLI and GUI	M,W,L	Yes	2012	Commercial	Energy, Power, DRAM
McPAT	[40]	CLI	L	Yes	2009	Research	Power, architecture
PAPI	[64]	CLI	L	Not tested	2010	Research	Power

Powmon is an experimental software that can be used to obtain power and performance counter information on mobile CPUs[4]. PAPI is an interface that is widely used in the community and provides an API to access performance counter information and also the specific RAPL interface registers to estimate energy and power consumption.

6.6 Energy estimation in machine learning

Estimation of energy consumption can be useful for machine learning experts for several reasons, as covered in Sections 6.3 and 6.4.5. In this section, we present an overview of the current research in machine learning regarding energy and power estimation with emphasis on the progress made in deep learning.

Machine learning models such as deep neural networks are characterized by parameters or weights that are used to transform input data into features. These models consist of two distinct phases of computation: the training phase and the inference phase. In the training phase, the deep neural network is designed, the number of layers, the size and type of each layer are selected, and weight parameters are learned. During the inference, the fixed weight parameters are tested on example input data to extract features from the input data. Current approaches for training these models mainly rely on desktop or server systems (including high-end GPUs, CPUs or FPGAs) to

[4]Please refer to : `http://www.powmon.ecs.soton.ac.uk/powermodeling/`

support the training of deeper models on large data sets. Meanwhile, the inference phase is typically performed on low-end embedded systems, for example, smart phones, wearables and others. However, a large body of research has emerged to optimize the energy-efficiency of these machine learning models and are driven by early energy modelling approaches applied in machine learning field [41].

One of the first models to estimate the energy of a neural network model was to count the number of multiply-accumulate (MAC) to model the number of floating-point operations for the CPU or the GPU and the number of weights to model the number of main memory accesses, of a pre-trained model, as proxies for energy [67]. Since the weights have to be loaded from DRAM that have high relative energy costs compared to a MAC operation, a large number of optimization efforts such as pruning [67], compression [68] and compact models [69], focused on reducing the number of weights or parameters of the neural network models.

Using the number of weights was deemed too simplistic and further attempts were made to improve the energy modelling approaches by incorporating an energy costs for different types data in different levels in the memory hierarchy. The authors in [41] built an energy estimation model by counting the number of times each data value is reused across the memory hierarchy using an optimization procedure. There are four-levels of memory hierarchy considered: DRAM, global buffer, array and register file, and the energy-cost for each level is extracted from a 65nm process. This energy estimation model is used to map an arbitrary shaped neural network model onto their application-specific accelerators such as Eyeriss. While the authors provide an energy estimation model to map the neural network model to their specific architecture design, details of their optimization procedure are not provided [2]. This methodology was later extended to prune weights during the training phase that account for higher energy-consumption [41].

Modelling the power at the application-level using performance counters for general-purpose processors have also begun to emerge. SyNERGY [3] presents a methodology to produce application-level energy measurements using models built from performance counters. The application tested is a simple inference based on convolutional neural networks on a single core of an ARM A57 CPU. They model the energy consumption of the convolutional layers at the system-level that incorporates the power used by the processor, memory and other peripherals. The performance counters chosen for the

study are the number of SIMD instructions executed and the number of main memory accesses. Since the convolutions are commonly restructured into matrix-matrix multiplications the assumption is that the two most important performance counter are the ones that contribute to computation and data movement. The instantaneous power is measured at the SoC level with the interactive Linux governor for power management and no change to the DVFS settings. These instantaneous power values are then converted to corresponding energy values. The authors initially built a simple multi-variable linear regression model to correlate the performance counter information with corresponding energy profile of the application. Finally, the authors extend the model to predict the number of SIMD instruction and bus accesses to the applications MAC count. This was done to relate the application-level features directly to the energy profile.

NeuralPower [4] applies a regression based approach to model the power consumption and the run-time for a desktop GPU. These models were built for the three main layers of a convolutional neural network, i.e., convolutional, pooling and fully-connected layers. They use real power and timing values to build predictive models for average power and runtime. The power values are obtained on an NVidia GTX1070 GPU using *nvidia-smi*. The authors use application-level features, such as, kernel size, number of layers and others as input for the power estimation model. They exclude the impact of voltage and frequency scaling by keeping the GPU in a fixed state. The output of the power predictive model and the runtime model are combined to give an estimate of energy-use per layer. Each of the per-layer estimates are subsequently added to get an estimate of the power consumption of the overall convolutional neural network. However, the authors do not provide the final prediction model to the reader and have a limited number of ConvNet models in the study.

DeLight [70] models the energy consumption of a simple feedforward neural network during its training phase. The authors model the energy-use in terms of basic arithmetic operations and communication of shared weights in a distributed training setting[5]. Two types of operation are modelled; multiply-add, which is a function of the number of connections between neurons of two adjacent layers, and the activation function, which is a scalar multiplication. The communication energy is a function of the number of shared weights. The coefficients for modelling these operations are obtained

[5]Sharing the weight parameter space among different cores

by running a micro-benchmarks on the CUDA cores of the Nvidia TK1 embedded platform.

Table 6.5 summarizes the areas in which energy estimation has been applied to deep learning. Most studies use energy estimation techniques during the inference phase and apply post-processing techniques to reduce the energy consumption of the neural network. While the general rule of thumb followed is to keep the count of the number of parameters as low to reduce the overhead in performance, very few studies have emerged to explicitly incorporate energy-use *a priori* during the training phase [70]. Approaches to automate the process of designing energy-aware neural network models use techniques such as reinforcement learning [71], Bayesian optimization [72] and genetic algorithms [73]. Furthermore, deep learning models are generally trained on desktop GPUs and energy estimation models covering GPUs will become necessary.

Table 6.5: *Areas where energy estimation has been applied in deep learning*

Paper	Training	Inference	CPU	GPU
[67]		X		X
[41]		X		
[3]		X	X	
[4]		X		X
[70, 72, 73],	X			X

6.7 Use cases

This section demonstrates two uses cases that estimate the energy at the application-level based on the classification in Section 6.3 and the technique based on performance counter information described in Section 6.4, for estimating the energy consumption in data mining and machine learning scenarios.

6.7.1 Data stream mining

Data stream mining algorithms build machine learning models online, as the data arrives, by reading the data only once. One of the main principles is that the data should not be stored, just the necessary statistics of the data. A typical scenario is a sensor or an IoT (Internet of Things) network producing a potentially infinite stream of data and the data stream mining algorithm building the model in real time. The state-of-the-art algorithms

in this area are able to handle changes in the input data distribution, known as concept drift [74].

The goal of this use case is to compare the energy consumption of two stream mining algorithms in two scenarios with and without concept drift. The first algorithm, the Very Fast Decision Tree (VFDT) [75], is able to build and update a decision tree in real time, but is not able to handle concept drift. The Hoeffding Adaptive Tree (HAT) [76] extends the VFDT to handle concept drift by changing the structure of the model to better approximate the new data. This use case investigates the extra energy cost needed to handle concept drift.

We have chosen to estimate the energy consumption using Intel Power Gadget, described in Section 6.5. This tool provides real-time estimations with almost no overhead. The datasets have been synthetically generated with MOA [77] (Massive Online Analysis) and the algorithms were run in the same platform. We use prequential evaluation, were instances are tested and then trained, giving online accuracy measurements. The chosen datasets are the random tree (no concept drift), and random RBF (Radial Basis Function, with concept drift, 0.001 speed change), both with 5 million instances. [6]

The results of running both algorithms 10 times and averaging the results are shown in Table 6.6. We can observe that HAT obtains higher accuracy for both datasets. For the concept drift dataset the HAT algorithm obtains 16% higher accuracy than the VFDT, at the cost of 1.12x more joules (11.8% higher energy consumption). On the other hand, for the random tree dataset, both algorithms obtain similar levels of accuracy (less than 0.5% difference) but the VFDT consumes significantly less energy compared to HAT. This is expected, since the extra computations performed by HAT to handle concept drift are not needed in this dataset.

Table 6.6: *Energy and accuracy of the VFDT and the HAT in concept drift and non-concept drift datasets with 5 million instances*

			Energy(J)		
Alg	Dataset	Acc(%)	Total	Processor	DRAM
HAT	RandomRBF(0.001)	**65.75**	578.03	558.34	19.69
HAT	RandomTree	**97.75**	758.80	728.32	30.49
VFDT	RandomRBF(0.001)	56.45	**516.92**	496.43	20.49
VFDT	RandomTree	97.40	**363.30**	348.60	14.70

[6] More information about the datasets is available here: `https://moa.cms.waikato.ac.nz/details/classification/streams/`

We can conclude that to obtain higher accurate models for concept drift scenarios comes at a cost of energy consumption. However, by doing an initial analysis of the energy consumed by the algorithm, the researcher or machine learning expert can make the adequate choices depending on the set of constraints. For instance, in a scenario with embedded devices where the battery is the main constraint, the researcher might choose to sacrifice that 16% of accuracy to extend the battery's life. Another option is to give the different alternatives to the user, providing them with the necessary information regarding energy efficiency and accuracy.

6.7.2 Convolutional neural network inference

An inference phase of a convolutional neural network (or ConvNet model) consists of passing an image through a pre-trained ConvNet. It undergoes a series of transformations of the input data into features that can be used to build image classifiers, object detectors and other computer vision applications. These transformations are arranged into layers such as Convolutional, pooling, normalization, fully-connected layers and others. Typically, these inferences are performed on mobile and embedded systems that have limited battery-life [5].

Our goals are i) to use an existing methodology to acquire per-layer energy of an example ConvNet, ii) apply a regression-based approach to performance counter information to predict convolutional layer energy, iii) validate with real energy measurements. We use the methodology proposed in SyNERGY [3], to find the functional-level or per-layer sections of the application by inserting code annotations in the Caffe C++ code. The code annotation library is provided by the ARM Streamline tool which is a pre-requisite to allow such fine-grained instrumentation of the code. The power is captured using the power sensor on the Jetson TX1 that provides system-level power. Similarly, performance counter information is captured using the ARM Streamline tool.

We use the regression model in [3], with coefficients for the number of SIMD instructions and number of bus accesses to be $x_1 = 3.34E-05$ and $x_2 = 3.18E-06$ respectively. We test this model on three example pre-trained ConvNet models, Inception-v3 [78], MobileNet [79] and DenseNet [80], as given in Table 6.7. We show both the predicted energy and actual measured energy for a single-threaded execution of ConvNet on an ARM Cortex-A57

from the Jetson TX1 board. We obtain an average relative accuracy[7] of 68.91% across the three test ConvNets. The predicted and measured energy consumption for the convolutional layers of MobileNet is lowest and they account for 95% of the computation within a deep convolutional neural network [5]. Therefore, given a choice between the three ConvNets that that MobileNet is the most energy-efficient ConvNet choice under similar execution environments.

Table 6.7: *Measured versus predicted energy consumption of the Convolutional layers of three example ConvNets on Cortex-A57*

ConvNet	Measured (mJ)	Predicted (mJ)	Relative Accuracy (%)
Inception-V3	10945.73	8073.55	73.70
MobileNet	2453.19	1546.80	63.05
DenseNet	8777.78	6148.16	70.00

6.8 Conclusions

Machine learning algorithms consume significant amounts of energy. However, the lack of evaluations based on energy consumption of these algorithms can be attributed to the lack of of appropriate tools to measure and build power models in existing machine learning suites, and because estimating energy consumption is a challenging task.

This paper addresses that challenge by presenting a review of the key approaches to estimate energy consumption from the computer architecture field, mapped to machine learning applications. We also describe the state-of-the-art methods to estimate energy consumption in particular for data mining and convolutional neural networks. Our synthesis of the surveyed papers provides the necessary guidelines to expose energy consumption methods to machine learning audiences interested in incorporating energy as metric in the design of machine learning systems. To demonstrate the usefulness of the synthesis, we present two use cases, which show, from the data mining and neural networks perspectives, how to apply the different estimation approaches. We show that the benefits of further research in energy estimations can help machine learning researchers gain significant insights when building machine learning systems.

Our survey also reveals the current state of energy estimations in machine learning. In particular, there are several works emerging to enable energy

[7] the absolute difference between actual and estimated energy over the actual energy

evaluations in machine learning either through energy prediction modelling as seen in NeuralPower [4] or by direct integrating power monitoring tools to existing machine learning suites as seen in SyNERGY [3]. However, current modelling approaches face another challenge; which is the rapid changes in neural network designs, implementations and hardware. Moreover, there is a fragmentation of the machine learning software ecosystem with multiple software suites and no common benchmarking suites. These modelling approaches will have to be adaptable to these changes and also be comprehensive. This is because most works target a few computational intensive layers with majority of the work confined to just convolutional neural networks.

Another area in which energy estimation is lacking are GPUs that are extensively used by machine learning researchers to train machine learning models. Few works such as NeuralPower [4] build desktop GPU-based energy estimation models while others target mobile CPUs and GPUs [3]. Future work will require further research to integrate the literature on GPU-based estimation methods applied for machine learning scenarios. Finally, there is a recent upsurge to build application specific hardware for machine learning [22]. To keep pace with both advancements in neural network designs and advances in the hardware community will require energy estimation models for these new emerging architectures either gathered on real systems or through neural network hardware simulators such as ScaleSIM [81]. As future work, we aim to investigate energy estimation approaches for GPUs and emerging application-specific hardware.

6.9 References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[2] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks”. In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138.

[3] C. F. Rodrigues, G. Riley, and M. Luján. “SyNERGY: An energy measurement and prediction framework for Convolutional Neural Net-

works on Jetson TX1". In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2018, pp. 375–382.

[4] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu. "NeuralPower: Predict and Deploy Energy-Efficient Convolutional Neural Networks". In: *Asian Conference on Machine Learning*. 2017, pp. 622–637.

[5] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar. "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices". In: *Proceedings of the 2015 international workshop on internet of things towards applications*. ACM. 2015, pp. 7–12.

[6] K. Gauen, R. Rangan, A. Mohan, Y.-H. Lu, W. Liu, and A. C. Berg. "Low-power image recognition challenge". In: *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE. 2017, pp. 99–104.

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `http://tensorflow.org/`.

[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).

[9] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. "Automatic differentiation in PyTorch". In: (2017).

[10] C. F. Rodrigues, G. Riley, and M. Luján. "Fine-grained energy profiling for deep convolutional neural networks on the Jetson TX1". In: *Workload Characterization (IISWC), 2017 IEEE International Symposium on*. IEEE. 2017, pp. 114–115.

[11] B. Goel, S. A. McKee, and M. Själander. "Techniques to measure, model, and manage power". In: *Advances in Computers*. Vol. 87. Elsevier, 2012, pp. 7–54.

[12] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma. "Modeling, profiling, and debugging the energy consumption of mobile devices". In: *ACM Computing Surveys (CSUR)* 48.3 (2016), p. 39.

[13] R. W. Ahmad, A. Gani, S. H. A. Hamid, F. Xia, and M. Shiraz. "A review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues". In: *Journal of Network and Computer Applications* 58 (2015), pp. 42–59.

[14] R. A. Bridges, N. Imam, and T. M. Mintz. "Understanding GPU power: A survey of profiling, modeling, and simulation methods". In: *ACM Computing Surveys (CSUR)* 49.3 (2016), p. 41.

[15] K. O'brien, I. Pietri, R. Reddy, A. Lastovetsky, and R. Sakellariou. "A survey of power and energy predictive models in HPC systems and applications". In: *ACM Computing Surveys (CSUR)* 50.3 (2017), p. 37.

[16] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, and V. Boeva. "How to Measure Energy Consumption in Machine Learning Algorithms". In: *ECML PKDD 2018 Workshops*. Ed. by C. Alzate, A. Monreale, H. Assem, A. Bifet, T. S. Buda, B. Caglayan, B. Drury, E. García-Martín, R. Gavaldà, I. Koprinska, S. Kramer, N. Lavesson, M. Madden, I. Molloy, M.-I. Nicolae, and M. Sinn. Cham: Springer International Publishing, 2019, pp. 243–255. ISBN: 978-3-030-13453-2.

[17] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[18] S. Kaxiras and M. Martonosi. "Computer architecture techniques for power-efficiency". In: *Synthesis Lectures on Computer Architecture* 3.1 (2008), pp. 1–207.

[19] C. Talarico, J. W. Rozenblit, V. Malhotra, and A. Stritter. "A new framework for power estimation of embedded systems". In: *Computer* 2 (2005), pp. 71–78.

[20] E. Garcia-Martin, N. Lavesson, and H. Grahn. “Identification of energy hotspots: A case study of the very fast decision tree”. In: *International Conference on Green, Pervasive, and Cloud Computing*. Springer. 2017, pp. 267–281.

[21] S. K. Rethinagiri, O. Palomar, R. Ben Atitallah, S. Niar, O. Unsal, and A. C. Kestelman. “System-level power estimation tool for embedded processor based platforms”. In: *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*. ACM. 2014, p. 5.

[22] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. “In-datacenter performance analysis of a tensor processing unit”. In: *arXiv preprint arXiv:1704.04760* (2017).

[23] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner. “Event-driven energy accounting for dynamic thermal management”. In: *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP’03)*. Vol. 22. 2003.

[24] C. Gilberto and M. Margaret. “Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events”. In: *ISLPED*. Vol. 5. 2005, pp. 8–10.

[25] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson. “Application-aware power management”. In: *Workload Characterization, 2006 IEEE International Symposium on*. IEEE. 2006, pp. 39–48.

[26] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. “Full-system power analysis and modeling for server environments”. In: International Symposium on Computer Architecture-IEEE. 2006.

[27] K. Singh, M. Bhadauria, and S. A. McKee. “Real time power estimation and thread scheduling via performance counters”. In: *ACM SIGARCH Computer Architecture News* 37.2 (2009), pp. 46–55.

[28] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. “Decomposable and responsive power models for multicore processors using performance counters”. In: *Proceedings of the 24th ACM International Conference on Supercomputing*. ACM. 2010, pp. 147–158.

[29] B. Goel, S. A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati. "Portable, scalable, per-core power estimation for intelligent resource management". In: *Green Computing Conference, 2010 International*. IEEE. 2010, pp. 135–146.

[30] V. Spiliopoulos, A. Sembrant, and S. Kaxiras. "Power-sleuth: A tool for investigating your program's power behavior". In: *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*. IEEE. 2012, pp. 241–250.

[31] M. J. Walker, A. K. Das, G. V. Merrett, and B. Hashimi. "Run-time power estimation for mobile and embedded asymmetric multi-core cpus". In: (2015).

[32] B. Goel and S. A. McKee. "A Methodology for Modeling Dynamic and Static Power Consumption for Multicore Processors." In: *IPDPS* (2016), pp. 273–282.

[33] A. Mazouz, D. C. Wong, D. Kuck, and W. Jalby. "An Incremental Methodology for Energy Measurement and Modeling". In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ACM. 2017, pp. 15–26.

[34] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett. "Accurate and stable run-time power modeling for mobile and embedded cpus". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.1 (2017), pp. 106–119.

[35] D. Brooks, V. Tiwari, and M. Martonosi. *Wattch: A framework for architectural-level power analysis and optimizations*. Vol. 28. 2. ACM, 2000.

[36] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. "The design and use of simplepower: a cycle-accurate energy estimation tool". In: *Proceedings of the 37th Annual Design Automation Conference*. ACM. 2000, pp. 340–345.

[37] R. Joseph and M. Martonosi. "Run-time power estimation in high performance microprocessors". In: *Proceedings of the 2001 international symposium on Low power electronics and design*. ACM. 2001, pp. 135–140.

[38] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria. “An instruction-level energy model for embedded VLIW architectures”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21.9 (2002), pp. 998–1010.

[39] B. C. Lee and D. M. Brooks. “Accurate and efficient regression modeling for microarchitectural performance and power prediction”. In: *ACM SIGOPS Operating Systems Review*. Vol. 40. 5. ACM. 2006, pp. 185–194.

[40] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. “McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures”. In: *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE. 2009, pp. 469–480.

[41] T.-J. Yang, Y.-H. Chen, and V. Sze. “Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning”. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 6071–6079.

[42] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee. “Instruction level power analysis and optimization of software”. In: *Technologies for wireless computing*. Springer, 1996, pp. 139–154.

[43] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. “RAPL: Memory power estimation and capping”. In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. Aug. 2010, pp. 189–194. DOI: 10.1145/1840845.1840883.

[44] Y. S. Shao and D. Brooks. “Energy characterization and instruction-level energy model of Intel’s Xeon Phi processor”. In: *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*. IEEE Press. 2013, pp. 389–394.

[45] M. A. Laurenzano, A. Tiwari, A. Jundt, J. Peraza, W. A. Ward, R. Campbell, and L. Carrington. “Characterizing the performance-energy tradeoff of small ARM cores in HPC computation”. In: *European Conference on Parallel Processing*. Springer. 2014, pp. 124–137.

[46] H. Mehta, R. M. Owens, and M. J. Irwin. “Energy characterization based on clustering”. In: *Proceedings of the 33rd annual Design Automation Conference*. ACM. 1996, pp. 702–707.

[47] M. D. Hill, J. R. Larus, A. R. Lebeck, M. Talluri, and D. A. Wood. "Wisconsin architectural research tool set". In: *ACM SIGARCH Computer Architecture News* 21.4 (1993), pp. 8–10.

[48] W.-T. Shiue and C. Chakrabarti. "Memory exploration for low power, embedded systems". In: *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*. ACM. 1999, pp. 140–145.

[49] Y. Zhang, R. Y. Chen, W. Ye, and M. J. Irwin. "System level interconnect power modeling". In: *ASIC Conference 1998. Proceedings. Eleventh Annual IEEE International*. IEEE. 1998, pp. 289–293.

[50] D. Burger and T. M. Austin. "The SimpleScalar tool set, version 2.0". In: *ACM SIGARCH computer architecture news* 25.3 (1997), pp. 13–25.

[51] K. Rajamani, H. Hanson, J. C. Rubio, S. Ghiasi, and F. L. Rawson. "Online power and performance estimation for dynamic power management". In: *IBM, RC-24007, Tech. Rep* (2006).

[52] M. Moudgill, J.-D. Wellman, and J. H. Moreno. "Environment for PowerPC microarchitecture exploration". In: *IEEE Micro* 19.3 (1999), pp. 15–25.

[53] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield. "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors". In: *IBM Journal of Research and Development* 47.5.6 (2003), pp. 653–670.

[54] K. Nose and T. Sakurai. "Analysis and future trend of short-circuit power". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19.9 (2000), pp. 1023–1030.

[55] Semiconductor Industries Association. *Model for Assessment of CMOS Technologies and Roadmaps (MASTAR)," 2005.*

[56] C. Auth, A. Cappellani, J.-S. Chun, A. Dalis, A. Davis, T. Ghani, G. Glass, T. Glassman, M. Harper, M. Hattendorf, et al. "45nm high-k+ metal gate strain-enhanced transistors". In: *VLSI Technology, 2008 Symposium on*. IEEE. 2008, pp. 128–129.

[57] J. L. Henning. "SPEC CPU2006 benchmark descriptions". In: *ACM SIGARCH Computer Architecture News* 34.4 (2006), pp. 1–17.

[58] E. E. Devices. *Watts up PRO*. 2009.

[59] C. Spearman. “The proof and measurement of association between two things”. In: *The American journal of psychology* 15.1 (1904), pp. 72–101.

[60] S. Eranian. “Perfmon2: a flexible performance monitoring interface for Linux”. In: *Proc. of the 2006 Ottawa Linux Symposium*. 2006, pp. 269–288.

[61] P. Guide. “Intel® 64 and IA-32 Architectures Software Developer’s Manual”. In: *Volume 3B: System programming Guide, Part* 2.14.9 (2011), pp. 14-31–14-40.

[62] S. Desrochers, C. Paradis, and V. M. Weaver. “A validation of DRAM RAPL power measurements”. In: *Proceedings of the Second International Symposium on Memory Systems*. ACM. 2016, pp. 455–470.

[63] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. “Measuring energy consumption for short code paths using RAPL”. In: *ACM SIGMETRICS Performance Evaluation Review* 40.3 (2012), pp. 13–17.

[64] D. Terpstra, H. Jagode, H. You, and J. Dongarra. “Collecting performance data with PAPI-C”. In: *Tools for High Performance Computing 2009*. Springer, 2010, pp. 157–173.

[65] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. “The scalable heterogeneous computing (SHOC) benchmark suite”. In: *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM. 2010, pp. 63–74.

[66] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout. “An Evaluation of High-Level Mechanistic Core Models”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* (2014). ISSN: 1544-3566. DOI: 10.1145/2629677.

[67] S. Han, J. Pool, J. Tran, and W. Dally. “Learning both weights and connections for efficient neural network”. In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[68] S. Han, H. Mao, and W. J. Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding”. In: *arXiv preprint arXiv:1510.00149* (2015).

[69] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[70] B. D. Rouhani, A. Mirhoseini, and F. Koushanfar. "Delight: Adding energy dimension to deep neural networks". In: *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM. 2016, pp. 112–117.

[71] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le. "Mnasnet: Platform-aware neural architecture search for mobile". In: *arXiv preprint arXiv:1807.11626* (2018).

[72] D. Stamoulis, E. Cai, D.-C. Juan, and D. Marculescu. "HyperPower: Power-and memory-constrained hyper-parameter optimization for neural networks". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 19–24.

[73] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, et al. "ChamNet: Towards Efficient Network Design through Platform-Aware Model Adaptation". In: *arXiv preprint arXiv:1812.08934* (2018).

[74] G. Hulten, L. Spencer, and P. Domingos. "Mining time-changing data streams". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 97–106.

[75] P. Domingos and G. Hulten. "Mining high-speed data streams". In: *Proc. 6th SIGKDD Int'l Conf. on Knowledge discovery and data mining*. 2000, pp. 71–80.

[76] A. Bifet and R. Gavaldà. "Adaptive learning from evolving data streams". In: *International Symposium on Intelligent Data Analysis*. Springer. 2009, pp. 249–260.

[77] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. "MOA: Massive Online Analysis". In: *Journal of Machine Learning Research* 11 (2010), pp. 1601–1604. URL: `http://portal.acm.org/citation.cfm?id=1859903`.

[78] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[79] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[80] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. “Densely connected convolutional networks.” In:

[81] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna. “Scale-sim: Systolic cnn accelerator”. In: *arXiv preprint arXiv:1811.02883* (2018).

7

Energy Efficiency Analysis of the Very Fast Decision Tree Algorithm

Eva García-Martín, Niklas Lavesson, Håkan Grahn

Abstract

Data mining algorithms are usually designed to optimize a trade-off between predictive accuracy and computational efficiency. This paper introduces energy consumption and energy efficiency as important factors to consider during data mining algorithm analysis and evaluation. We conducted an experiment to illustrate how energy consumption and accuracy are affected when varying the parameters of the Very Fast Decision Tree (VFDT) algorithm. These results are compared with a theoretical analysis on the algorithm, indicating that energy consumption is affected by the parameters design and that it can be reduced significantly while maintaining accuracy.

7.1 Introduction

Data stream mining is gaining importance with the evolution of hardware, sensor systems and technology. The rate at which data is generated is increasing day by day, challenging storage and computational efficiency [1]. Digital Universe Study [2] has predicted that by 2020, 40,000 exabytes of data will be processed, most of them originating from devices that automatically generate data. Many algorithms in data stream mining are designed to process fast and potentially infinite streams [3, 4].

Traditionally, the machine learning community has considered accuracy as the main factor when building algorithms. With the appearance of big data analytics and data stream mining, scalability has also been a key factor to consider. In this context, scalability stands for how fast an algorithm can process the incoming data. The problem that we address in this study is

the fact that few researchers in the data mining community consider energy consumption as an important measure.

It has been shown that energy consumption can be reduced in every layer of the Open Systems Interconnection (OSI) model [5, 6]. Hardware solutions to reduce energy consumption have been focused on, e.g. using the Dynamic Voltage Frequency Scaling (DVFS) technique and on parallel computing [7, 8]. During recent years, the interest in developing energy efficient software solutions has increased significantly, leading to a creation of applications to measure energy consumption in software [9].

This paper introduces energy consumption and energy efficiency as important factors to consider during data mining algorithm analysis and evaluation, and to demonstrate the use of these factors in a data stream mining context. The consideration of energy efficiency can help companies and researchers move towards green computing [10] while improving the business profits.

Social networks is a very good example of a domain in need of efficient data processing and analysis of algorithms. Companies such as Facebook, Twitter, and Instagram, rely on large data clusters consuming vast amounts of energy. Facebook, for example, generates 4 million posts every minute [11], which creates interesting challenges for algorithms that are running continuously in their network. One of these challenges is to optimize such algorithms in terms of energy consumption. Even a small improvement will reduce energy on a large scale, due to the nature of the network.

We conducted an experiment to illustrate a possible scenario where energy consumption is relevant to study. More specifically, we studied how energy and accuracy are affected by changing the parameters of the VFDT (Very Fast Decision Tree) algorithm [3]. We make a comparison between the theoretical analysis on the algorithm and the experimental results, which indicate that it is possible to significantly reduce the energy consumption of the VFDT algorithm while maintaining similar levels of accuracy. The main contribution of this paper is the introduction of energy consumption as a key factor to consider in data mining algorithms. This is supported by a theoretical and empirical analysis that illustrate an example on how to build sustainable and efficient algorithms and the reasons behind energy consumption.

This paper is an extension of the paper titled: *Energy Efficiency in Data*

Stream Mining [12]. While such publication centers on observing the energy consumption only from an empirical perspective, this study motivates the experimental setup and results by analyzing the behavior of the VFDT from a theoretical and empirical perspective. Therefore, we can compare how the algorithm behaves in reality from what we predicted theoretically. On top of that, more relevant parameters have been chosen and two real world datasets have been added to the experiment.

7.2 Background

In this section we first explain the importance of energy consumption in data mining. Then, we briefly explain data stream mining and why it is different from standard data mining, and finally we introduce some terminology related to power, energy, energy efficiency and computational efficiency.

7.2.1 Energy-awareness

The demand for energy is increasing day by day [5]. World leaders and scientists focus on finding a solution towards this problem, centering on two key factors: developing new sources of clean energy and decreasing energy usage [13] [14], which would lead to a reduction in CO_2 emissions. The main reason why researchers and every citizen should be aware of energy consumption is because energy pollutes. Every device that we use in a daily bases that consumes energy produces CO_2. Nowadays, based on a study conducted by the World Health Organization, air pollution kills more people than malaria and aids combined [15]. This argument is based on what is known as ecological or environmental footprint [16], that measures how much impact a certain person or action has in relation to the environment. For instance, carbon footprint measures how many greenhouse gases are produced by an individual or event, expressed as CO_2 [17]. Therefore, if companies and individuals are aware of the footprint of their computations, their impact could be reduced by making them energy efficient.

There have been studies that measure the environmental impact of queries in search engines [18]. Considering that there are approximately 66k Google queries per second [19], reducing the CO_2 emissions of search queries will significantly impact the environment. If we translate this example to data stream mining, we can picture the execution of data stream mining

algorithms in servers running during 24 hours a day, for a complete year. In this case, building energy-aware algorithms has the following consequences:

- Reduction of CO_2 emissions to the atmosphere.
- Reduction of air pollution, therefore reducing the number of deaths per year due to this matter.
- Reduction of the money spent on energy.
- Increase of the battery life of mobile devices and sensor networks, if the algorithm is implemented in such contexts.

7.2.2 Data Stream Mining

Data stream mining is the process of building models by exploring and extracting patterns from a stream of data.

The core assumption of data stream mining, in comparison to data mining, is that the examples are inspected only once, so we have to assume that if they are not processed immediately they are lost forever [20, 21]. Moreover, it is considered that the data arrives online, with no predefined order, at a high-speed and with time-changing characteristics. Data stream mining algorithms should be able to process potentially infinite streams while updating the model incrementally [4, 22].

7.2.3 Terminology

In this section we clarify several concepts related to energy, power, and efficiency. Energy is a measurement of the amount of *fuel* used for a specific application. It is measured in Joules (J) or kWh. Power is a measurement of the rate at which energy is consumed. It is measured in Joules/second, which is equal to Watts (W). The following is an example that illustrates the relationship between power and energy: A process is running for 3.94 seconds consuming an estimate power of 1.81 W. The total energy consumed is: $3.94 \times 1.8 = 7.092J = Ws = 1.99 \times 10^{-3}$ Wh.

Energy efficiency has a specific definition at Green500 [23], being, *The amount of operations per watt a computer can perform.* This definition is related to hardware. In this study, whenever we mention energy efficiency we refer to reducing the energy consumption of some process or algorithm.

In theoretical machine learning, researchers introduced the computational learning theory [24], where they analyze the computational complexity of algorithms. They approach computational efficiency as a way of designing less computationally complex algorithms that can run in polynomial time.

7.3 Related Work

In this section we first review literature related to energy awareness in software and hardware. Then, we examine relevant work in the data stream mining field, focusing on the VFDT algorithm. Finally, we review papers that are related to both energy consumption and data stream mining.

Research in energy awareness at the software level started many years ago, when researchers began to realize the importance of the energy consumed by a software application. In 1994, the first systematic attempt to model the power consumption of the software components of a system was presented [25]. After that, in 1999, PowerScope was presented [26], a software tool for profiling the energy usage of applications. The novelty of this approach is that energy consumption can be mapped to program structure to analyze which procedures consume more energy. Companies such as Microsoft [27] and Intel [28] have invested in developing software tools to help developers reduce the energy consumption of their applications. During the past years, the Spiral research group [29] has gained interest in building energy efficient software. They show that energy consumption depends not only on the time and number of computations, but also on the stress of the processor, the I/O operations and many other factors. They have developed a software tool, PowerAPI, where they show that they can get high accurate modeling of the power consumption of software applications. They have evaluated their model by comparing their results and method with hardware power meters obtaining promising results [9, 30].

In relation to energy efficiency at the hardware level, one of the most important techniques, implemented in most contemporary processors, is Dynamic Voltage Frequency Scaling (DVFS). DVFS is a power saving technique used and improved by many researchers. One improvement is Real Time DVFS, an implementation of DVFS for real time systems [7]. Another area that is gaining importance nowadays is parallel computing, where there are relevant energy savings by employing more cores on a processor [8]. Several energy-saving approaches, such as *Cost optimization for power-aware*

computing have been developed in the past years [5].

In relation to data stream mining, researchers have developed efficient approaches to mine data streams, as outlined below. There have been several reviews conducted in data stream mining since 2005. Two general reviews [1, 31], portray techniques and concepts such as data-based techniques, task-based techniques, data stream classification and frequent pattern mining. More specific reviews center on topics such as sensor networks [32] and knowledge discovery [22].

From the reviews explained above, we have extracted six main techniques and approaches in data stream mining: Data stream clustering [33], Data stream classification [3], Frequent Pattern Mining [34], Change Detection in data streams [35, 36], Sliding window techniques [37] and Stream mining in sensor networks [32, 38]. We have decided to focus in Data Stream classification and change detection in data streams.

Concept drift refers to a change between the input data and the target variable on an online supervised learning scenario. The first framework that dealt with concept drift was proposed to also address efficiency and robustness [39]. Nowadays, researchers consider concept-drift as an important aspect when building algorithms for other specific purposes. A survey on different methods that address concept drift has been conducted in 2014 [40].

Classification is considered a challenging problem in a data stream mining scenario [31]. The main reason is that many of the traditional classification techniques and algorithms were designed to build models from static data.

One of the key breakthroughs in supervised online learning was made with the development of the Hoeffding Tree algorithm and the Very Fast Decision Tree (VDFT) learner [3]. In contrast to previous algorithms, such as as SPRINT [41] and ID5R [42], this new approach was able to deal with potential infinite streams, arriving at a fast pace and with low computational cost. The VFDT learner is able to process examples at a high rate in constant time. One year later, the same authors created a new version of the VDFT algorithm, CVFDT, that was able to adapt to concept-drift [4]. Another extension on the VFDT algorithm appeared two years later, with a new decision tree learner that could efficiently process numerical attributes [43]. In the same line, a decision tree algorithm was created for spatial data streams [44]. We would like to mention relevant methods that address different classification problems, namely: On-Demand classification [35, 45],

Online Information Network (OLIN) [46], LWClass [47], ANNCAD [48], and SCALLOP [49].

In relation to energy awareness in data stream mining, several researchers have conducted studies where they emphasize the importance of energy consumption [1, 50, 51]. While the first two are concerned on energy savings for sensor networks, the second one centers on examine the energy consumption of different data analysis techniques. To the best of our knowledge, the last work is the one most related to ours.

We can observe that there is no specific research on making energy consumption a key factor on data stream mining, since the research has been centered towards specific applications or hardware modifications. We would like to change this approach by proposing energy consumption as the new factor to consider when building, optimizing or creating new algorithms in data stream mining. We believe that this is the next natural step to take, since other researchers in similar fields, hardware and software, have already taken that step.

7.4 Theoretical Analysis

This section aims to theoretically analyze the behavior of the Very Fast Decision Tree (VFDT) algorithm [3]. VFDT is an online decision tree algorithm able to build a decision tree from a stream of data by analyzing the data sequentially and only once.

The decision tree is built sequentially, where the tree waits until it gathers enough examples or instances from the stream. After those n instances arrive, the algorithm analyzes them and obtains the best attribute to split the tree on. The key feature is to obtain the optimal value of n that will split in the same attribute as if we had all examples available to analyze. To obtain the first best value of n, the authors make use of the Hoeffding Bound [52], represented by ϵ in Equation 7.1.

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{7.1}$$

This bound states that with probability 1-δ, the chosen attribute at a specific node after seeing n number of examples, will be the same attribute as if the algorithm had seen infinite number of examples. Therefore, δ

represents one minus the probability of choosing the correct attribute to split on. The reason is that there will be no split on a certain attribute unless $\Delta\overline{G} > \epsilon$. $\Delta\overline{G}$ stands for the difference in information gain between the best two attributes. Thus, if the number of examples n is small, ϵ will be high, making it harder to split on an attribute unless $\Delta\overline{G}$ is big enough, meaning that there is a clear attribute that is the winner. Based on the equation, whenever we see more examples, n increases, making ϵ smaller and then making it easier to split on the top attribute. The reasoning behind this is that whenever we see more examples we are more confident on the split. In order to speed up the computations, some parameters are introduced, that will make the algorithm behave in a slightly different way that the one currently explained.

The next paragraphs theoretically analyze how accuracy and energy would differ when varying the different parameters of the VFDT algorithm. The chosen parameters to be varied are: *nmin*, τ, δ, *memory limits*, *memory management*, *split criterion* and *poor attributes removal*. As a general observation, the theoretical analysis made about the parameters can not be generalized to all cases, since it would vary depending on the input data. For that reason, the assumptions that are made in the following paragraphs, are based on the reasoning and experiments from the original paper by the authors.

nmin parameter is the minimum number of examples that the algorithm must see before calculating ϵ to check if there are sufficient statistics for a good split. The authors introduce this parameter to reduce execution time and computational effort when building the tree, since it is very unlikely that after just one instance the algorithm has a more convincing split. The default value of *nmin* is 200. If the value of *nmin* increases, then accuracy will be slightly reduced, since the tree will have a lower number of nodes. From the original paper, the difference in accuracy was of a merely 1.1%, but the execution time was 3.8 times faster when using and increasing *nmin*. Therefore, we predict that the energy would decrease when increasing *nmin*, since we would be decreasing the computational effort and time to analyze each instance.

τ parameter represents the tie breaking parameter. Whenever the difference between both attributes is small enough, that means both attributes are equally good, making no sense to wait a longer time for more examples to make a split. The absence of this parameter has been shown to decrease

accuracy, since the decision tree contains fewer nodes in it. However, being able to make more splits on the data allows to obtain a finer grained decision tree. In an extreme situation where both attributes are exactly the same, the tree would stall, failing to grow. So increasing τ could, in an ideal scenario with an infinite stream of data, increase accuracy and decrease energy, since the tree is built before, reducing the time and the number of computations [21]. But if we analyze the same amount of examples, then increasing τ could increase the energy, due to the fact that with lower τ we make less number of computations.

δ parameter represents one minus the probability of choosing the correct attribute to split on. If δ increases then the desired probability is smaller. Hence, the tree will grow faster, having more nodes. Since the difference on nodes between a higher and a lower δ will not be as high as when removing τ, and the probability of a correct split is lower, our assumption is that the accuracy will be lower when δ increases. At the same time, if δ decreases, then the probability of making a correct split increases, increasing accuracy. In terms of energy, we believe that the power and time consumed to build the tree will vary depending on the incoming data. We hypothesize that with a lower δ there will be more power spent on computing information gain rather than in building nodes. However we currently do not have the knowledge of which consumes more power. At the same time, the tree could be built faster since we are spending less time on building the tree nodes.

In terms of *memory limits* and *memory management*, we predict that with a lower memory limit, the tree will induce fewer nodes, having less energy consumption and less accuracy. At the same time, the tree uses pruning techniques to reduce the memory spent on building the tree, removing less promising leaves, which in some cases could improve accuracy. It will depend on how different is the memory limit and how many nodes differ from each setting to correctly predict if the accuracy will be lower or higher. In a realistic case we assume that limiting the memory consumption and the tree growth will decrease accuracy and energy.

The default split criterion used by the authors in this algorithm is *information gain*. We have tested also *Gini index*. From a theoretical aspect, it has been shown [53] that it is not conclusive which of the two criterion will perform a better job in general, since they both have shown similar results, differing only by 2%.

The last parameter that is going to be modified is *removing poor attributes*.

This parameter aims to analyze attribute performance to find attributes that perform poorly and which are very unlikely to be chosen for a further split. The process that the authors follow is by analyzing the information gain of all attributes in every split, and when this value, for a specific attribute, is less than the information gain of the best attribute by more than a difference of ϵ, then the attribute is ignored for that leaf. In theory, this method should increase accuracy and decrease the amount of computations.

Table 7.1 represents a summary of all the predictions of the parameters variations explained above. It shows how energy, accuracy and tree size will vary when increasing or decreasing the mentioned parameters.

Table 7.1: *Summary of the theoretical behavior of the parameters*

PARAMETER	MODIFICATION	ACC	ENERGY	NODES
nmin	Increase	Lower	Lower	Fewer
	Decrease	Higher	Higher	More
τ	Increase	Higher	Higher	More
	Decrease	Lower	Lower	Fewer
δ	Increase	Lower	†	More
	Decrease	Higher	†	Fewer
MEM1	100KB	Lower	Lower	Fewer
	2GB	Higher	Higher	More
MEM2	ON	Lower	Lower	Fewer
SPLT CRIT	S2	†	†	†
RPA	ON	Higher	Lower	Fewer

†=The variation depends on the input data.

7.5 Experimental Design

7.5.1 Problem Definition

In order to empirically study the different parameters of the VFDT algorithm we have created an experiment where we vary the parameters theoretically analyzed in Section 7.4. This experiment aims to evaluate the performance of the algorithm under different setups in terms of accuracy, energy, execution time and size of the tree. An implicit goal is understanding why varying the parameters in a certain way increases or decreases accuracy and energy, and if it matches with the theoretical reasoning. The experiment has three phases. First, we obtain the datasets, then we input them into the algorithm

under different setups, and we finally evaluate the performance of each model in terms of accuracy and energy consumption. The way to measure energy is explained in Section 7.5.3.

7.5.2 Data Gathering

We have gathered four different datasets to perform this experiment. Two datasets are synthetically generated and the other two are real world datasets. The main difference between synthetic and real world datasets, is that the first ones are randomly generated based on a specific function and distribution, and the second ones are representations of some measure that exists in reality. The idea is to show that the solution proposed in this paper of analyzing energy consumption of algorithm, applies to both real world and controlled environment. It improves the generalizability of the results.

The synthetic datasets have been generated with MOA (Massive Online Analysis) [54], and the functions: *Random Tree Generator*, *Hyper Plane generator*. The random tree function generates a tree as explained by the authors of the VFDT algorithm [3]. We have chosen this dataset because is the same dataset that the authors of the VFDT use in their experiments, so we consider it as a baseline of a standard behavior of the algorithm. Then we chose a more challenging synthetic dataset, since the hyperplane generator is often used to test algorithms that can handle concept drift, such as CVFDT. Even though this algorithm is not developed to handle concept drift, we wanted to test the different setups in a completely different dataset than the random tree. The Hyper plane generator uses a function to generate data in the form of a plane in d dimensions [4]. The orientation of the hyperplane can easily be varied by adjusting its weights, creating different concepts. Depending on the coordinates of the plane, the examples are labeled as negative or positive. All synthetic generators have generated a total of 1 million instances, and we have chosen the number of numerical and nominal attributes based on the default settings in MOA. The tree-generated dataset is a binary classification dataset with 5 nominal and 5 numerical attributes. The hyperplane-generated dataset is also a binary classification dataset with 10 different attributes and 2 concept drift attributes.

Since usually synthetic datasets do not have the same properties as real world datasets, we have decided to add two real world datasets to the experiment. The first one represents instances that try to predict good poker

hands based on a given hand, available from the MOA official website [55]. There are a total of 1,025,010 instances and 11 attributes and has been normalized by the MOA researchers. The second real world dataset is the normalized airline dataset, created by Elena Ikonomovska [56]. This classification dataset classifies flights into delayed or not depending on the route of the flight, based on the departure and arrival airports. It contains a total of 8 attributes, being: Airline, flight number, origin, destination, day of the week, elapsed time, duration of the flight, and if there was a delay or not. The motivation behind these datasets is the amount of instances available, making them perfect candidates to test data stream mining algorithms. Table 7.2 summarizes the information from the different datasets, regarding the number of instances, attributes and type of the dataset.

Table 7.2: *Datasets summary*

Dataset	Name	Type	Instances	Nominal attributes	Numeric attributes
1	Random tree	Synthetic	1,000,000	5	5
2	Hyperplane	Synthetic	1,000,000	0	10
3	Poker	Real world	1,025,010	5	5
4	Airlines	Real world	539,383	4	3

7.5.3 Methodology

This section aims to explain the settings of the experiment, the parameters varied and the tools used to perform it.

7.5.3.1 Parameter choice

We have chosen to vary the parameters explained in Section 7.4, namely: *nmin*, τ, δ, *memory limits*, *memory management*, *split criterion*, and *removing poor attributes*. *nmin* was varied from the default value, 200, to a maximum value of $1,700$ with steps of 500. τ was varied from the default value, 0.05, to a maximum value of 0.13, a minimum value of 0.01 and with steps of 0.04. δ was varied from the default value, 10^{-7}, to a maximum and minimum values of 10^{-1} and 10^{-10}, respectively. The step is of 10^{-3}. Memory limit varied from 100KB, to 30MB (default value) until 2GB, that was the maximum allowed by MOA (Massive Online Analysis), the tool that will be furthered explained. The memory management and removing poor attributes were tested by activating and deactivating them. Finally, Gini

index was tested against Information Gain. Every parameter was varied while maintaining the other parameters constant, in their default value. The aim is to understand the behavior of each parameter on its own, without having external interference with the rest of the parameters. A summary of the parameters setup is shown in Table 7.3.

Table 7.3: *Parameter Configuration Index. Different configurations of the VFDT algorithm.*

IDX	$nmin$	τ	δ	MEM1	MEM2	S.CRT	RPA
A	200	0.05	10^{-7}	30MB	No	S1	No
B	**700**	0.05	10^{-7}	30MB	No	S1	No
C	**1,200**	0.05	10^{-7}	30MB	No	S1	No
D	**1,700**	0.05	10^{-7}	30MB	No	S1	No
E	200	**0.01**	10^{-7}	30MB	No	S1	No
F	200	**0.09**	10^{-7}	30MB	No	S1	No
G	200	**0.13**	10^{-7}	30MB	No	S1	No
H	200	0.05	$\mathbf{10^{-1}}$	30MB	No	S1	No
I	200	0.05	$\mathbf{10^{-4}}$	30MB	No	S1	No
J	200	0.05	$\mathbf{10^{-10}}$	30MB	No	S1	No
K	200	0.05	10^{-7}	**100KB**	No	S1	No
L	200	0.05	10^{-7}	**2GB**	No	S1	No
M	200	0.05	10^{-7}	30MB	**Yes**	S1	No
N	200	0.05	10^{-7}	30MB	No	**S2**	No
O	200	0.05	10^{-7}	30MB	No	S1	**Yes**

MEM1=Memory limits. MEM2=Memory management. S.CRT=Split criterion. S2=Gini index. RPA=Removing poor attributes.

7.5.3.2 Procedure

There will be a total of 15 parameter combinations for every dataset, indexed from A-O. Since there are a total of 4 datasets, the number of executions will be 60. Every execution represents the choice of applying one algorithm, with a specific parameter tunning, on one of the datasets. In parallel, we will be measuring how much energy is the execution consuming. Each combination of dataset and parameter tunning has been computed a total of ten times, to then obtain the average and standard deviation of all of them. The average of such computations are the results portrayed in the next section. A summary of these configurations is shown in Table 7.4. The experiment was carried out in a Linux machine with a 2.70 GHz Intel i7 processor (four cores), and with 8 GB of RAM. The models built from analyzing the synthetic generated

data where trained and tested on 1 million instances. For the testing phase, new randomly generated data was used. On the other hand, for the real world datasets, the testing was performed on the same data as the training phase

Table 7.4: *Design summary*

	Quantity	Type
Datasets	4	Random Tree generator, Hyperplane generator, Poker, Airlines
Measures	6	Time, Power, Energy, Accuracy, Number of nodes, Tree depth
Parameter configuration	15	Represented in Table 7.3, as: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O
Executions	10	10 executions for every parameter configuration on each dataset

7.5.3.3 Tools

We need to differentiate between two tools. The first tool, MOA (Massive Online Analysis), is used to execute the VFDT with the different parameter settings. Running in parallel to MOA is PowerAPI [30], a tool developed by the Spirals Research team [29], which is able to measure how much power different processes are consuming. PowerAPI [30] has been successfully tested by the authors [9] to compute the differences in terms of energy between some software process in different laptops. The energy is calculated by integrating the power consumed from the process during the execution time.

7.5.4 Evaluation

The last step of the experiment is the evaluation process. In order to evaluate the different settings on the different datasets, we have chosen four measures. The first two are accuracy and energy. We want to discover if there is a trade-off between energy and accuracy, i.e. we will only obtain a lower energy consumption by reducing the accuracy. Or, on the other hand, if there are specific setups were we can reduce energy consumption without loosing accuracy, i.e. smart setups. From the theoretical analysis on the algorithm we have observed that a possible relationship with accuracy is with the number of nodes of the tree. Therefore, the last two evaluation measures considered are the number of nodes (size) and the depth of the tree.

7.6 Results and Analysis

7.6.1 General Analyses

Table 7.5: *Experimental results. The best accuracy and energy results for each dataset are highlighted.*

S	1 T	P	E	A	N	D	2 T	P	E	A	N	D
A	4.44	8.59	38.10	96.91	1,134	8	5.85	7.39	43.19	90.93	655	12
B	3.90	7.32	28.57	96.31	661	8	5.65	7.12	40.22	90.99	607	11
C	3.90	7.10	27.71	96.24	570	7	5.66	7.75	43.90	91.22	575	10
D	3.82	6.99	**26.67**	95.91	495	7	5.57	7.77	43.31	90.98	515	11
E	4.46	8.44	37.63	95.57	699	8	5.36	9.15	49.08	90.57	57	7
F	4.74	7.35	34.84	97.93	1,541	9	7.42	6.89	51.10	90.38	2,071	19
G	5.09	7.30	37.11	97.94	2,074	11	9.25	6.87	63.55	89.87	3,863	18
H	5.20	7.55	39.21	**98.27**	2,181	12	9.51	6.78	64.45	89.75	3,971	19
I	4.58	7.90	36.22	97.54	1,403	9	6.32	7.01	44.32	90.82	1,129	12
J	4.29	8.18	35.10	96.35	888	8	5.73	7.58	43.48	**91.24**	471	11
K	3.98	7.37	29.31	95.43	1,134	8	5.03	7.60	**38.20**	84.89	655	12
L	4.40	8.32	36.60	96.91	1,134	8	5.92	7.17	42.48	90.93	655	12
M	4.39	7.95	34.91	96.91	1,134	8	5.91	7.12	42.11	90.93	655	12
N	6.12	6.67	40.79	83.11	1,735	117	5.77	7.60	43.90	90.72	619	11
O	4.34	7.85	34.08	96.91	1,134	8	5.83	7.24	42.23	90.93	655	12

S	3 T	P	E	A	N	D	4 T	P	E	A	N	D
A	7.29	7.33	53.48	76.63	297	16	6.54	9.17	59.96	67.31	8,582	4
B	7.03	8.29	58.27	70.67	181	11	6.09	8.41	51.20	67.01	7,984	3
C	7.25	8.57	62.16	79.44	195	16	6.06	8.40	50.91	67.01	8,228	4
D	7.70	8.76	67.45	73.66	149	16	5.99	8.19	**49.01**	66.65	7,895	3
E	8.36	7.49	62.60	73.74	149	13	6.19	8.81	54.50	67.09	5,127	3
F	7.14	7.46	53.26	82.56	791	17	6.94	8.46	58.74	67.77	12,307	3
G	7.46	7.53	56.25	84.88	1,285	22	7.32	8.41	61.56	**67.92**	14,137	3
H	6.87	7.06	**48.49**	**93.06**	1,991	20	7.29	8.43	61.46	67.90	14,001	4
I	7.32	7.36	53.88	78.93	575	18	6.75	8.61	58.14	67.70	10,871	3
J	7.89	8.98	70.89	68.71	119	13	6.41	9.14	58.58	66.97	6,618	3
K	7.26	7.43	53.92	76.63	297	16	6.56	8.96	58.74	67.31	8,582	4
L	7.19	7.57	54.41	76.63	297	16	6.51	8.88	57.82	67.31	8,582	4
M	7.34	7.61	55.85	76.63	297	16	6.60	9.47	62.48	67.31	8,582	4
N	8.99	7.25	65.17	37.83	1,380	102	7.62	9.24	70.37	58.77	1,490	53
O	7.18	7.37	52.95	76.63	297	16	6.52	9.06	59.07	67.31	8,582	4

T(s)=Time in seconds. P(W)=Power in Watts. E(J)=Energy in Joules.
A(%)=Percentage of correctly classified instances. N=Nodes. D=Depth.

Table 7.5 shows the results obtained from the experiment, for each dataset and measuring energy, time, power, number of nodes, depth of the tree and accuracy. First, we compare energy and accuracy, to understand if there is a visible trade-off between the increase of energy and the increase of accuracy. We can observe from Figure 7.1 how there seems to be a linear relationship between the increase of energy and the decrease of accuracy. Apparently,

based on datasets 1, 3, and 4, whenever the energy increases the accuracy decreases. This result is promising in terms of our ultimate goal, trying to make energy efficient algorithms. Since there is no trade-off between energy and accuracy, a sacrifice of accuracy is not needed to develop energy efficient algorithms. In all datasets there is one outlier, that presents a lower accuracy than the rest, this is the parameter split criterion set to Gini index. This configuration presents a significantly lower levels of accuracy without reducing energy consumption in comparison with the other parameters.

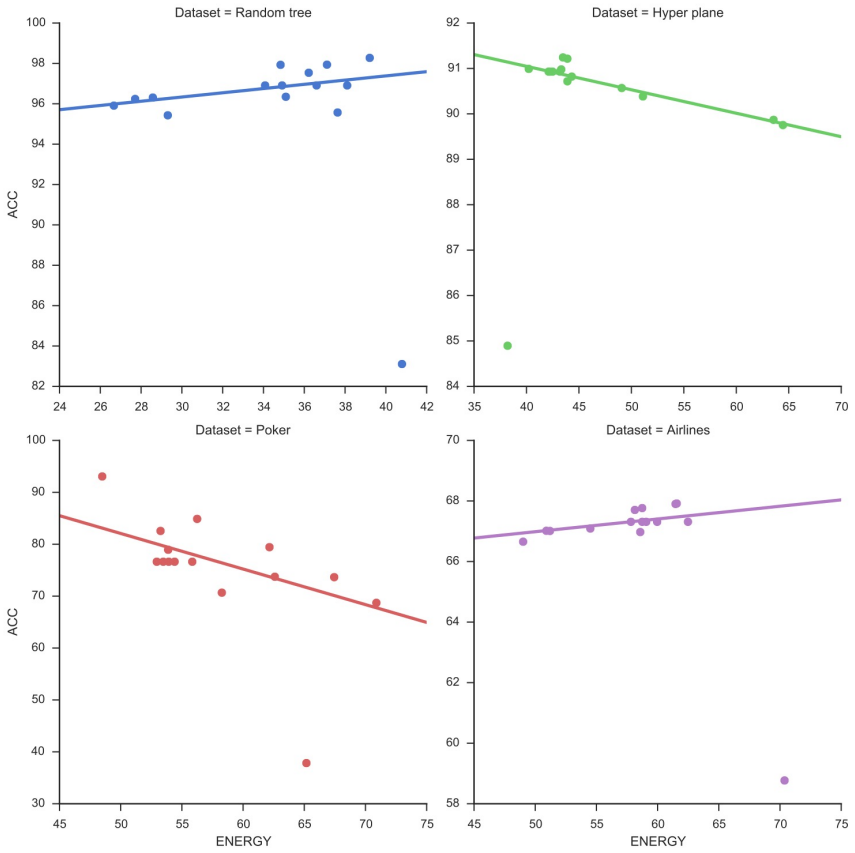

Figure 7.1: *Scatter plot with a linear relationship between accuracy (ACC) and energy (Joules) for every dataset.*

Figure 7.2 shows how energy and accuracy vary for the different parameter configurations. Although this energy variation between parameters is analyzed in depth in the next subsection, we believe it is relevant to mention

that energy differs significantly between specific configurations. For instance, for the third dataset, energy and accuracy vary for every single parameter, suggesting the relevancy of measuring energy consumption and not leaving the development choices to pure chance.

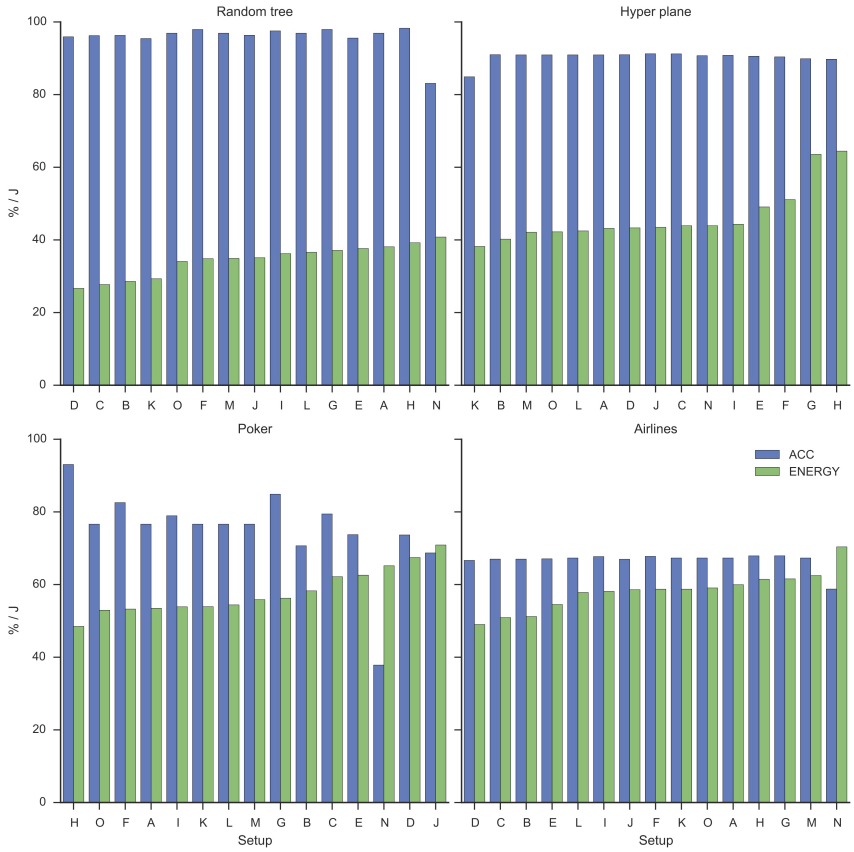

Figure 7.2: *Barplot showing the energy and accuracy variations for every parameter configuration on each dataset. Accuracy (ACC) is measured in percentage of correctly classified instances and energy is measured in Joules.*

In this paragraph we analyze the relationship between the number of nodes and accuracy, portrayed in Figure 7.3. As has been explained in the theoretical analysis section, some parameter configuration can increase accuracy although intuitively it should be decreased. This is the case of the parameter τ. When τ is increased, intuitively the accuracy should decrease, since you are allowing splits on *not so good attributes*. However, although

at first we predicted a decrease in accuracy, since there are significantly more nodes in the tree, the accuracy increases. This can be observed from Tables 7.5 and 7.6. The main reason is that with a significant increase of nodes, the tree is able to represent the data in a more fine-grained way, thus increasing accuracy. If we zoom in the highest values for each dataset, we can observe that for datasets 1, 3, and 4, there seems to be a higher accuracy whenever the number of nodes is higher.

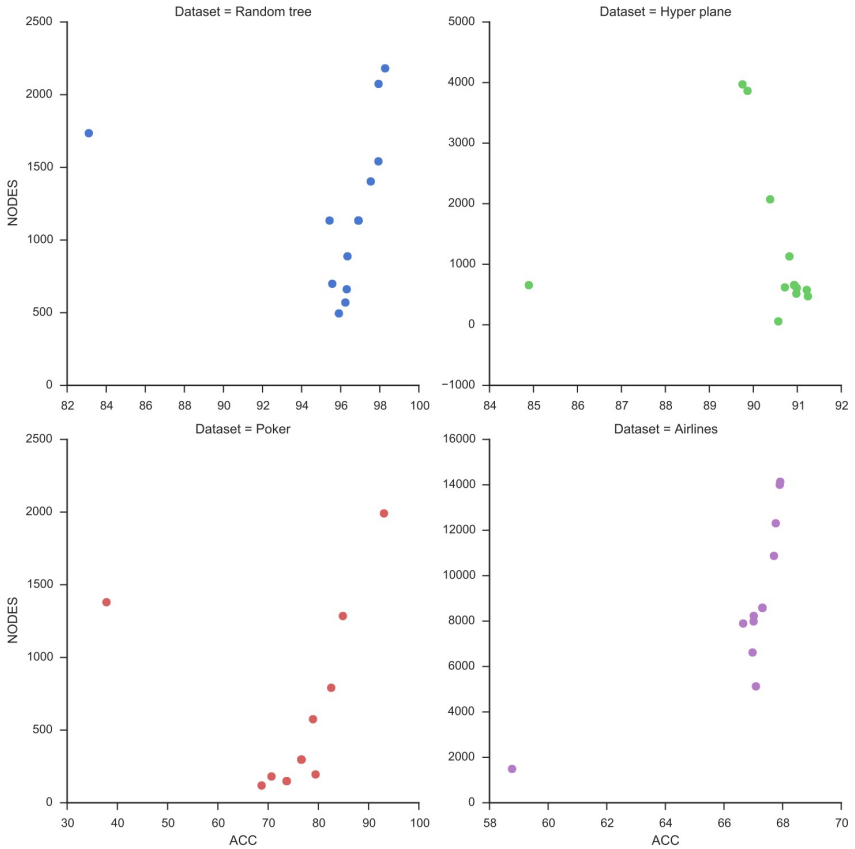

Figure 7.3: *Scatter plot with a linear relationship between accuracy and nodes for every dataset.*

A final analysis is related to the number of nodes and the energy consumed. When we look at the data from Table 7.5, whenever energy increases, time or power increases (since it is the product of both). The interesting measure is to see whether is the power or the time the one causing this energy

increase. We have observed from Figure 7.4 that whenever the number of nodes of a tree increase, so does the execution time. This phenomenon occurs more clearly for datasets 1, 2, and 4, suggesting that there is a relationship between time and nodes. Therefore, a conclusion is the increase of energy is related to an increase on time, probably due to an increase in the number of nodes.

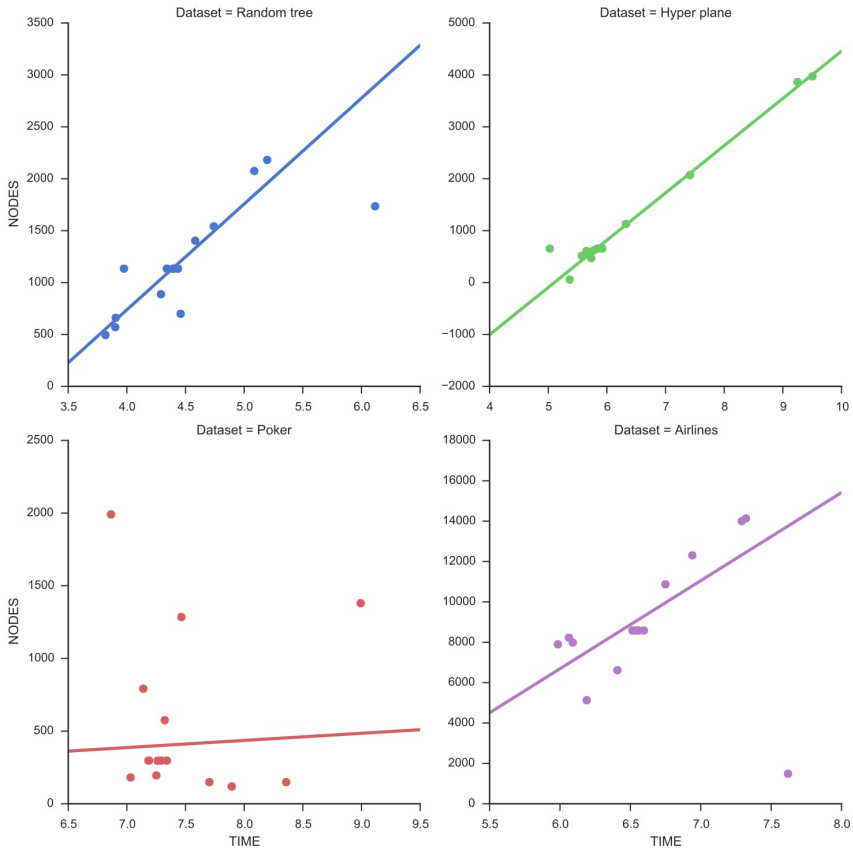

Figure 7.4: *Scatter plot with a linear relationship between time (seconds) and nodes for every dataset.*

A second measurement is power. We have observed that the power is not linearly correlated with the number of nodes. Figure 7.5 suggests that whenever the number of nodes is higher, the power is lower, higher or the same. We have not encountered a variable that is directly correlated with the increase of power, so we will analyze its behavior for each parameter in

the next subsection.

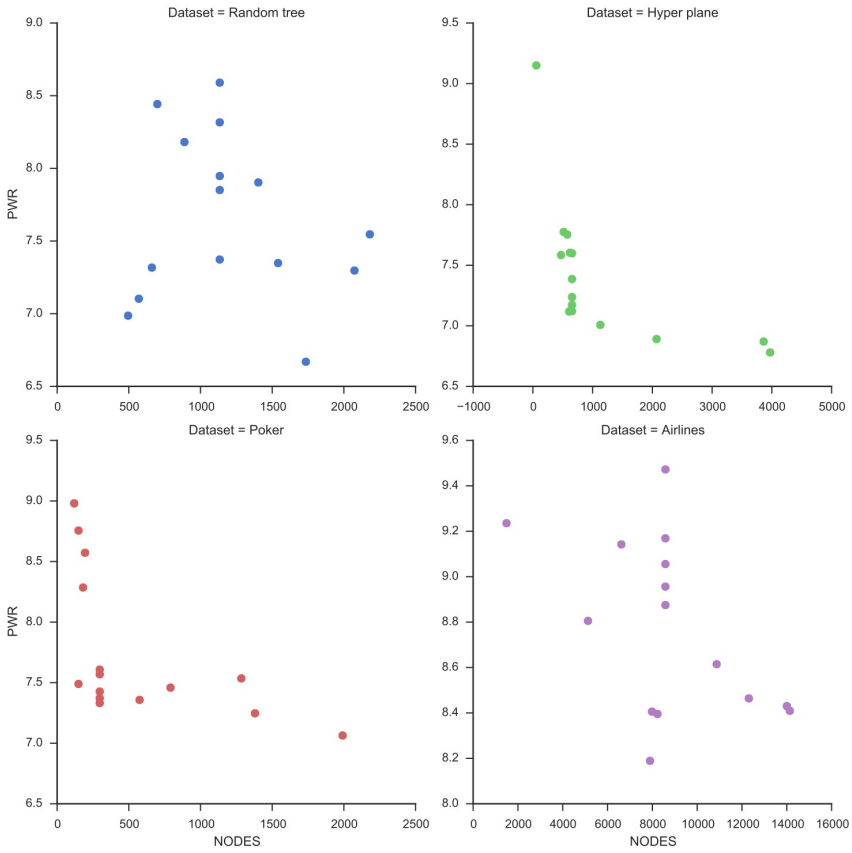

Figure 7.5: *Scatter plot with a linear relationship between power (W) and nodes for every dataset.*

7.6.2 Parameter Analysis

This section aims to compare the parameters' behavior from the theoretical predictions with the empirical results. For that, we have created Table 7.6, that shows the summary of the real behavior of the parameters obtained in the experiment.

Table 7.6: *Summary of the real behavior of the parameters in the experiment. This table portrays how energy, accuracy and the number of nodes vary when increasing, decreasing or modifying certain parameters of the VFDT algorithm.*

		nmin	τ	δ	MEM1		MEM2	S2	RPA
		↑	↑	↓	2GB	100KB			
D1	ACC	Dec	Inc	Dec	‡	Dec	‡	Dec	‡
	ENG	Dec	‡	Dec	Dec	Dec	Dec	Inc	Dec
	NDS	Few	More	Few	‡	‡	‡	More	‡
D2	ACC	‡	Dec†	Inc	‡	Dec	‡	‡	‡
	ENG	‡	Inc	Dec	Dec	Dec	Dec	Inc	Dec
	NDS	Few	More	Few	‡	‡	‡	Few	‡
D3	ACC	†	Inc	Dec	‡	‡	‡	Dec	‡
	ENG	Inc	Dec†	Inc	Inc	‡	Inc	Inc	Dec
	NDS	Few	More	Few	‡	‡	‡	More	‡
D4	ACC	‡	Inc	Dec	‡	‡	‡	Dec	‡
	ENG	Dec	Inc	Dec	Dec	‡	Inc	Inc	Dec
	NDS	Few†	More	Few	‡	‡	‡	Fewer	‡

†=It is not a constant decrease or increase throughout the configurations. Inc=Increase. Dec=Decrease. Few=Fewer nodes. More=More nodes. ACC=Accuracy. ENG=Energy. NDS=Nodes. ‡=There is no variation.

7.6.2.1 Parameter *nmin*

Observing the behavior of the *nmin* parameter across all datasets, we see that the accuracy is not significantly affected by this parameter, there is some decrease in dataset 1, but only of a 1%. In terms of power, however, we can see an important variation of Watts in datasets 1, 3, and 4. Both datasets 1 and 4 have an increase of power when *nmin* is increased. We checked what both datasets have in common to see the reasons behind this increase. The only characteristic that we find in common is that when power decreases, the depth of the tree is lower and also small in comparison to the other datasets. Another reason for this decrease in power when *nmin* increases is because we are computing less times the value of ΔG, therefore saving power. In terms of energy, when *nmin* increases, energy decreases and increases depending on the dataset. For datasets 1 and 4 it decreases, as was predicted in the theoretical section, for dataset 2 is stable and for dataset 3 it increases. In terms of nodes, we predicted that with higher *nmin* the number of nodes will decrease, and it is exactly what happened in all datasets. Finally, looking at time, in general for all datasets except

for the third one, time decreases when *nmin* increases, which is reasonable since we are looking into less batches of data, therefore the tree is computed faster.

7.6.2.2 Parameter τ

When the value of τ increases, so does the accuracy for datasets 1, 3, and 4. Dataset 2 experiences a non-stable accuracy value when increasing τ. Dataset 3 has an increase in accuracy of 11%, mainly due to the increase in the number of nodes, increasing around 2,000 nodes. That is also the case for all datasets, they significantly increase their number of nodes when τ increases, making this the main reason, from our understanding, for the accuracy increase. It matches with what we theoretically predicted, and with the results shown in the original paper of the authors [3]. In terms of energy, it varies depending on the parameter value. For datasets 2 and 4, there is a significant increase in energy. Dataset 3 experiences an important decrease in energy and dataset 1 does not vary energy significantly. Datasets' 2 and 4 increase of energy is due to a significant increase in time, and a slight reduction in power. Maybe this increase in time is due to the time that it takes to build more nodes. Dataset 3 decreases energy because it decreases time, while power varies between the setups. We predicted that energy would increase, matching with the behavior of datasets 2 and 4.

7.6.2.3 Parameter δ

When δ decreases, the probability of making a correct split increases. In this case, datasets 1 and 3 experience a significant decrease in accuracy. Especially dataset 3, that varies its accuracy a 25% (from $\delta = 10^{-1}$ to $\delta = 10^{-10}$). Comparing δ from the default value to the highest value ($\delta = 10^{-1}$) , accuracy increases from 76.6% to 93%. Intuitively, it should be the opposite, that increasing δ would decrease accuracy, since there would be less confidence of making a correct split. However, since the number of nodes increases significantly, being almost 7 times more nodes for the value $\delta = 10^{-1}$, then accuracy increases by that much. In all datasets, when δ decreases, the number of nodes decreases also, as we predicted, since there are less splits. Taking a look into the energy aspect, when δ increases energy significantly decreases for datasets 1 and 2. Dataset 4 does not vary energy significantly and dataset 3 increases energy when δ decreases. We discovered

that when $\delta = 10^{-1}$ for dataset 3, not only do we get a 25% increase in accuracy, as shown before, but we also get a decrease in energy of 22J.

7.6.2.4 Parameter MEM1

When the memory for the tree is restricted to 100KB, accuracy decreases by 1% for the first dataset and by 6% for the second dataset. For the other two datasets the accuracy stays the same, suggesting that the tree did not need more memory than 100KB. The number of nodes does not vary either in this case. The same happens when the memory is set to 2GB, the accuracy is the same across all datasets and the number of nodes does not vary at all. The reason the number of nodes is the same is because the implementation deactivates nodes when the memory is limited, but it does not remove them. In terms of power, time and energy, for the first two datasets there is a significant decrease of energy when the memory limit is set to 100KB. In the first dataset, the decrease is due to both a decrease in time and a decrease in power, which makes sense since the algorithm needs to analyze less number of nodes, consuming less power and taking less time. This is not the case for the two last datasets, since we already mentioned that probably the tree is not making use of such parameter, keeping energy at similar levels. When this parameter is set to the value of 2GB, the energy decreases slightly across all datasets except for the third one.

7.6.2.5 Parameter MEM2

When the parameter memory management (MEM2) is active, the tree stops growing when the memory limit is hit. The achieved accuracy and the number of nodes are exactly the same across all datasets. Energy does vary across all datasets, although it is not a big difference. Datasets 1 and 2 have a decrease in energy, while datasets 3 and 4 have an increase in energy. These variations are both due to the change of power and time, except for the second dataset, where there is a decrease on power but an increase on time.

7.6.2.6 Parameter S2

When the split criterion is set to *Gini index*, there is a significant decrease of accuracy across all datasets except for the second one. It is interesting to notice that even though it creates higher number of nodes for the first

and third datasets, the accuracy is still significantly lower. In the second dataset the accuracy is maintained and the number of nodes decreased. In terms of energy, there is an increase of energy across all datasets, although power and time vary in a different way in all of them, in most of them when this parameter is chosen the time to build the tree increases. Based on these results, we would not recommend to choose this splitting criterion. However, we have not investigated the reasons behind these results, therefore we suggest to do that in future work studies.

7.6.2.7 Parameter RPA

When the parameter *removing poor attributes* is chosen, we expect to have a higher accuracy and a decrease in energy. From the experiment we can observe that accuracy is maintained across all datasets, and energy is decreased across also all datasets, although the decrease is not significant in all of them. This increase is mostly due to a slightly decrease in time and a decrease in power. The nodes are maintained for all datasets, which is not what we predicted. The reason could be that even though the tree is using less attributes, the amount of instances to analyze is the same.

7.6.2.8 Summary of the parameter analysis

In general, we can observe that tunning the parameters in a different way outputs different values of energy consumption and accuracy. At the same time, it depends on the type of dataset which parameters will give better results in terms of energy consumption. For instance, while for the first dataset, the set of *nmin* to 1,700 will give the best results, for the third dataset it gives very poor results in terms of energy consumption. The reason behind this is that the VFDT algorithm will create different models, different decision trees, depending on the input data. A general observation is that the third dataset, the poker dataset, behaves in a different way in comparison to the other three datasets. On the same line, setting *nmin* to 700 seems to give good results across all datasets except the third one, what suggests that is a good option for future researchers. From looking at Table 7.1, we conclude that, in general, *removing poor attributes* has a positive impact on energy consumption without affecting accuracy. Also, decreasing the value of δ, decreases the energy consumption and accuracy, but the accuracy decrease is not significant.

7.7 Conclusions and Future Work

The aim in this paper is to introduce energy consumption as an important factor during data mining algorithm evaluation and analysis. While performance and computational effort are factors usually considered in data mining, energy consumption is seldom evaluated. Energy awareness leads to reducing CO_2 emissions, increasing battery life of mobile devices and reducing air pollution.

In order to understand the impact of taking energy consumption into consideration, we have analyzed the behavior in terms of energy and accuracy of the VFDT (Very Fast Decision Tree) algorithm when modifying certain parameters. First, we theoretically analyzed how increasing or decreasing certain parameters of such algorithm would affect the tree structure, the accuracy and the energy consumed. Then, we created an experiment where we empirically vary the same parameters of the VFDT algorithm under four different datasets. We have compared the empirical with the theoretical results, and found that there is indeed a significant variation in terms of energy consumption that depends on how the algorithmic setup is designed. The results also indicate that it is possible to significantly reduce the energy consumption of an algorithm without reducing accuracy by varying correctly the parameters of the algorithm.

Future work is to investigate why certain parameter choices consume more energy than others. For this purpose, we aim to break down data stream mining algorithms into generic sub tasks to allow a more fine-grained comparison of energy consumption across various algorithms and algorithm configurations. Finally, we plan to obtain more challenging real world datasets to test how energy can vary on these type of datasets.

7.8 References

[1] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. “Mining data streams: a review”. In: *ACM Sigmod Record* 34.2 (2005), pp. 18–26.

[2] *Digital Universe Study*. `https://www.emc.com/collateral/analyst-reports/idc-digital-universe-united-states.pdf`.

[3] P. Domingos and G. Hulten. "Mining high-speed data streams". In: *Proc. 6th SIGKDD Int'l Conf. on Knowledge discovery and data mining*. 2000, pp. 71–80.

[4] G. Hulten, L. Spencer, and P. Domingos. "Mining time-changing data streams". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 97–106.

[5] C. Reams. "Modelling energy efficiency for computation". PhD thesis. University of Cambridge, 2012.

[6] H. Zimmermann. "OSI reference model–The ISO model of architecture for open systems interconnection". In: *IEEE Transactions on Communications* 28.4 (1980), pp. 425–432.

[7] P. Pillai and K. G. Shin. "Real-time dynamic voltage scaling for low-power embedded operating systems". In: *ACM SIGOPS Operating Systems Review*. Vol. 35. 5. ACM. 2001, pp. 89–102.

[8] J. Li and J. F. Martı´nez. "Power-performance considerations of parallel computing on chip multiprocessors". In: *ACM Transactions on Architecture and Code Optimization (TACO)* 2.4 (2005), pp. 397–422.

[9] A. Noureddine, R. Rouvoy, and L. Seinturier. "Monitoring energy hotspots in software". In: *Automated Software Engineering* 22.3 (2015), pp. 291–332.

[10] A. Hooper. "Green computing". In: *Communication of the ACM* 51.10 (2008), pp. 11–13.

[11] *How Much Data is Generated Every Minute on Social Media?* `http://wersm.com/how-much-data-is-generated-every-minute-on-social-media/`. Published: August 19,2015.

[12] E. García Martín, N. Lavesson, and H. Grahn. "Energy Efficiency in Data Stream Mining". In: *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*. ACM. 2015, pp. 1125–1132.

[13] S. Chu. "The energy problem and Lawrence Berkeley National Laboratory". In: *Talk given to the California Air Resources Board* (2008).

[14] "Address of the president, Lord Rees Of Ludlow OM KT PRS, given at the anniversary meeting on 1 December 2008". In: *Notes and Records of the Royal Society of London* 63.2 (2009), pp. 183–190. ISSN: 00359149.

[15] M. Naghavi, H. Wang, R. Lozano, A. Davis, X. Liang, M. Zhou, S. E. V. Vollset, A. Abbasoglu Ozgoren, R. E. Norman, T. Vos, et al. "Global, regional, and national age–sex specific all-cause and cause-specific mortality for 240 causes of death: a systematic analysis for the Global Burden of Disease Study 2013". In: *The Lancet* 385.9963 (2015), pp. 117–171.

[16] *Ecological Footprint*. https://en.wikipedia.org/wiki/Ecological_footprint.

[17] *Carbon Footprint*. https://en.wikipedia.org/wiki/Carbon_footprint.

[18] *Search Queries*. http://searchengineland.com/calculating-the-carbon-footprint-of-a-google-search-16105.

[19] *Google Search Stats*. http://searchengineland.com/calculating-the-carbon-footprint-of-a-google-search-16105.

[20] A. Rajaraman and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[21] A. Bifet and R. Kirkby. "Data stream mining a practical approach". In: (2009).

[22] J. Gama. *Knowledge discovery from data streams*. CRC Press, 2010.

[23] *Green 500*. www.green500.org.

[24] M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

[25] V. Tiwari, S. Malik, and A. Wolfe. "Power analysis of embedded software: a first step towards software power minimization". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2.4 (1994), pp. 437–445.

[26] J. Flinn and M. Satyanarayanan. "PowerScope: A tool for profiling the energy usage of mobile applications". In: *WMCSA '99 Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*. IEEE. 1999, pp. 2–10.

[27] *Empowering Developers to Estimate App Energy Consumption*. http://research.microsoft.com/apps/pubs/default.aspx?id=166288.

[28] *Energy Efficient Software Development*. `https://software.intel.com/en-us/energy-efficient-software`.

[29] *Spirals Research Group*. `https://team.inria.fr/spirals/`.

[30] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier. "Powerapi: A software library to monitor the energy consumed at the processlevel". In: *ERCIM News* (2013).

[31] C. C. Aggarwal. *Data streams: models and algorithms*. Vol. 31. Springer Science & Business Media, 2007.

[32] M. M. Gaber. "Data stream processing in sensor networks". In: *Learning from Data Streams*. Springer, 2007, pp. 41–48.

[33] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. "Clustering data streams". In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE. 2000, pp. 359–366.

[34] R. Agrawal, T. Imieliński, and A. Swami. "Mining association rules between sets of items in large databases". In: *Acm sigmod record*. Vol. 22. 2. ACM. 1993, pp. 207–216.

[35] C. C. Aggarwal. "A framework for diagnosing changes in evolving data streams". In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM. 2003, pp. 575–586.

[36] D. Kifer, S. Ben-David, and J. Gehrke. "Detecting change in data streams". In: *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. 2004, pp. 180–191.

[37] M. Datar, A. Gionis, P. Indyk, and R. Motwani. "Maintaining stream statistics over sliding windows". In: *SIAM Journal on Computing* 31.6 (2002), pp. 1794–1813.

[38] M. Garofalakis, J. Gehrke, and R. Rastogi. "Querying and mining data streams: you only get one look a tutorial". In: *SIGMOD Conference*. 2002, p. 635.

[39] H. Wang, W. Fan, P. S. Yu, and J. Han. "Mining concept-drifting data streams using ensemble classifiers". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2003, pp. 226–235.

[40] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. "A survey on concept drift adaptation". In: *ACM Computing Surveys (CSUR)* 46.4 (2014), p. 44.

[41] J. Shafer, R. Agrawal, and M. Mehta. “SPRINT: A scalable parallel classifier for data mining”. In: *Proc. 1996 Int. Conf. Very Large Data Bases*. Citeseer. 1996, pp. 544–555.

[42] P. E. Utgoff. “Incremental induction of decision trees”. In: *Machine learning* 4.2 (1989), pp. 161–186.

[43] R. Jin and G. Agrawal. “Efficient decision tree construction on streaming data”. In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2003, pp. 571–576.

[44] Q. Ding, Q. Ding, and W. Perrizo. “Decision Tree Classification of Spatial Data Streams Using Peano Count Trees”. In: *Proceedings of the 2002 ACM Symposium on Applied Computing*. SAC ’02. Madrid, Spain: ACM, 2002, pp. 413–417.

[45] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. “On demand classification of data streams”. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 503–508.

[46] M. Last. “Online classification of nonstationary data streams”. In: *Intelligent Data Analysis* 6.2 (2002), pp. 129–147.

[47] M. M. Gaber, S. Krishnaswamy, and A. Zaslavsky. “On-board mining of data streams in sensor networks”. In: *Advanced methods for knowledge discovery from complex data*. Springer, 2005, pp. 307–335.

[48] Y.-N. Law and C. Zaniolo. “An adaptive nearest neighbor classification algorithm for data streams”. In: *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer. 2005, pp. 108–120.

[49] F. Ferrer-Troyano, J. S. Aguilar-Ruiz, and J. C. Riquelme. “Discovering decision rules from numerical data streams”. In: *Proceedings of the 2004 ACM symposium on Applied computing*. ACM. 2004, pp. 649–653.

[50] J. Gama and M. M. Gaber. *Learning from data streams*. Springer, 2007.

[51] R. Bhargava, H. Kargupta, and M. Powers. “Energy consumption in data analysis for on-board and distributed applications”. In: *Proceedings of the ICML*. Vol. 3. 2003, p. 47.

[52] W. Hoeffding. "Probability inequalities for sums of bounded random variables". In: *Journal of the American statistical association* 58.301 (1963), pp. 13–30.

[53] L. E. Raileanu and K. Stoffel. "Theoretical comparison between the gini index and information gain criteria". In: *Annals of Mathematics and Artificial Intelligence* 41.1 (2004), pp. 77–93.

[54] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. "MOA: Massive Online Analysis". In: *Journal of Machine Learning Research* 11 (2010), pp. 1601–1604. URL: http://portal.acm.org/citation.cfm?id=1859903.

[55] *MOA Datasets*. http://moa.cms.waikato.ac.nz/datasets/.

[56] *Elena Ikonomovska Airline Dataset*. http://kt.ijs.si/elena_ikonomovska/data.html.

8

Identification of Energy Hotspots: A Case Study of the Very Fast Decision Tree

Eva García-Martín, Niklas Lavesson, Håkan Grahn

Abstract

Large-scale data centers account for a significant share of the energy consumption in many countries. Machine learning technology requires intensive workloads and thus drives requirements for lots of power and cooling capacity in data centers. It is time to explore green machine learning. The aim of this paper is to profile a machine learning algorithm with respect to its energy consumption and to determine the causes behind this consumption. The first scalable machine learning algorithm able to handle large volumes of streaming data is the Very Fast Decision Tree (VFDT), which outputs competitive results in comparison to algorithms that analyze data from static datasets. Our objectives are to: (i) establish a methodology that profiles the energy consumption of decision trees at the function level, (ii) apply this methodology in an experiment to obtain the energy consumption of the VFDT, (iii) conduct a fine-grained analysis of the functions that consume most of the energy, providing an understanding of that consumption, (iv) analyze how different parameter settings can significantly reduce the energy consumption. The results show that by addressing the most energy intensive part of the VFDT, the energy consumption can be reduced up to a 74.3%.

8.1 Introduction

Current advancements in hardware together with the availability of large volumes of data, have inspired the field of machine learning into developing state-of-the-art algorithms that can process these volumes of data in real-time. For that reason, many machine learning algorithms are being implemented in big data platforms and in the cloud [1]. Examples of such applications

are Apache Mahout and Apache SAMOA [2], frameworks for distributed and scalable machine learning algorithms.

There are several desired properties for an algorithm to handle large volumes of data: processing streams of data, adaptation to the stream speed, and deployment in the cloud. The Very Fast Decision Tree (VFDT) algorithm [3] is the first machine learning algorithm that is able to handle potentially infinite streams of data, while obtaining competitive predictive performance results in comparison to algorithms that analyze static datasets. Since these algorithms often run in the cloud, an energy efficient approach to the algorithm design could significantly affect the overall energy consumption of the cluster of servers.

The VFDT and other streaming algorithms are only evaluated in terms of scalability and predictive performance. The aim of this paper is to profile the Very Fast Decision Tree algorithm with respect to its energy consumption and to determine the causes behind this consumption. Our objectives are to: (i) establish a methodology that profiles the energy consumption of decision trees at the function level, (ii) apply this methodology in an experiment with four large datasets (10M examples) to obtain the energy consumption of the VFDT, (iii) conduct a fine-grained analysis of the functions that consume most of the energy, providing an understanding of that consumption, (iv) analyze how different parameter settings can significantly reduce the energy consumption. We have identified the part of the algorithm that consumes the most amount of energy, i.e. the energy hotspot. The results suggest that the energy can be reduced up to a 74.3% by addressing the energy hotspot and by parameter tuning the algorithm.

The paper is organized as follows. In Section 8.2 we give a background explanation of decision trees and the VFDT, continuing with a review of related work. In Section 8.3 we explain the proposed method to profile energy consumption and how to apply it to the VFDT. Sections 8.4 and 8.5 present the experiment and the results and analysis of the experiment. Finally, we present the conclusions and pointers to future work in Section 8.6.

8.2 Background

8.2.1 Decision Trees and Very Fast Decision Tree (VFDT)

In a classification problem, we have a set of examples in the form (x, y). x represents the features or attributes, and y represents the label to be predicted. The goal is to find the function, or model, that predicts y given x ($y = f(x)$) [3]. Decision trees are a common type of algorithms used in machine learning that represent f in the form of a tree. A node in the tree represents a test on an attribute, and the branches of such node, known as literals, represent the attribute values. The leaves represent the labels y. When the model is built, in order to predict the label of a new example x_i, the example passes through the nodes based on the different attribute values, until it reaches a leaf. That leaf will be the label y. In order to build the model, the algorithm uses a divide-and-conquer approach. The dataset is passed to the first node, and based on that data chunk, the attribute with the highest information gain is chosen as the root node. The dataset is then **split** based on that attribute choice, and each chunk of data is passed to the corresponding child. The process is repeated recursively on each node, until the information is **homogeneous** enough, then the leaf is **labeled** with the appropriate class.

Very Fast Decision Tree (VFDT) [3] is a decision tree algorithm that builds a tree incrementally. The data is analyzed sequentially and only once. The algorithm analyzes the first n instances of the data stream, and chooses the best attribute as the root node. This node is updated with the literals, each being a leaf. The following examples will be passed to the next leaves and follow the same procedure of replacing leaves by decision nodes. On each iteration, the statistics on each leaf are updated with the new example values. This is done by first sorting an example to a leaf l, based on the attribute values of that example and on the parent nodes, and then updating the times each attribute value is observed in l. After a minimum number of examples n are seen in l, the algorithm calculates the two attributes with the highest information gain (G). Let $\Delta G = G(X_a) - G(X_b)$ be the difference between the information gain of both attributes. If $\Delta G > \epsilon$ the leaf is substituted by an internal node with the attribute with highest G. ϵ represents the Hoeffding Bound [4], shown in Eq. 8.1. This bound states that the chosen attribute at a specific node after seeing n number of examples, will be the same attribute as if the algorithm had seen infinite number of

examples, with probability 1-δ.

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{8.1}$$

8.2.2 Related Work

This section focuses on two areas. The first area regarding energy efficiency in computing and the second relating machine learning, big data and energy efficiency.

Many energy-aware hardware solutions have been implemented. For instance, the Dynamic Voltage Frequency Scaling (DVFS) power saving technique is used in many contemporary processors. Several energy-saving approaches present energy efficient solutions for computation [5]. Regarding green computing at the software level, several publications [6–8] address the importance of developing energy-aware solutions, applications and algorithms. One of the key points is that still there is a very abstract-level research that aims at making energy efficient computations. Companies such as Google, Microsoft, and Intel, are committed to build software, hardware and data centers that are sustainable, energy efficient and environmentally friendly[1]. The Spirals[2] research group builds energy efficient software, showing different factors that affect energy consumption at the processor level [9].

Moving on to machine learning, there have been different approaches and studies to evaluate machine learning algorithms on large scale datasets. We have identified three studies that we consider to be the most relevant in terms of large scale experiments that follow a fine-grained analysis. The first comparison between different algorithms on large scale datasets was conducted in terms of predictive performance [10] in 1995. This study empirically compares 17 machine learning algorithms across 12 datasets using time and accuracy. More than a decade later, a study was conducted that empirically compared ten supervised learning algorithms across 11 different datasets [11] by using eight accuracy-based performance measures. While these studies analyzed the total or average algorithm performance, another study was presented that evaluated algorithms in terms of time, by using a more detailed approach [12]. More specifically, the authors empirically

[1] http://www.theverge.com/2016/7/21/12246258/google-deepmind-ai-data-center-cooling

[2] https://team.inria.fr/spirals/

compare eight machine learning algorithms for time series prediction and with respect to accuracy and the computation time for every function per time series. In the past years there has been an increase in designing machine learning algorithms in distributed systems that are able to analyze big data streams [2, 13]. The Vertical Hoeffding Tree (VHT) [14] algorithm has been recently implemented to extend the VFDT. It is the first distributed streaming algorithm for learning decision trees. There is also a different perspective on how to use machine learning to make cloud computing environments more energy efficient [15].

There is a current increase of interest on energy efficiency algorithm design starting from the deep learning community, where they try to reduce the overall consumption of a neural network by pruning several nodes in the different layers while minimizing the error [16]. In the field of data stream mining, although they have published several algorithms that can handle large amounts of data, such as the VFDT [3] and it's extensions, we have yet to find an empirical evaluation of these algorithms with respect to energy consumption at the same detailed level. We believe that reducing the energy consumption of these kind of algorithms will have a significant impact in the overall consumption of data centers. In a previous work [17], we evaluated the impact on energy consumption of tuning the parameters of the VFDT. This work was extended [18] to choose more relevant parameters that could impact energy consumption based on a theoretical analysis of such parameters. For this paper, we focus on investigating the causes behind the energy consumption, by doing a fine-grained analysis of the energy consumption at the function level of the same algorithm.

8.3 Energy Profiling of Decision Trees

In order to analyze the energy consumption of the VFDT, we present a methodology to profile the energy consumption of decision tree learners at the function level. This approach allows us to: identify the energy hotspot of the algorithm, by discovering which functions are consuming most of the energy; and compare the energy consumption of different algorithms of the same class. The goal is to break down a specific algorithm into its specific functions, to then map them to the generic functions of the same algorithm class. We apply this to the VFDT in the following subsections. The method is divided in the following four steps:

1. Identification of the generic functions of a decision tree.
2. Identification of the specific functions of the algorithm to be profiled.
3. Mapping the specific functions of step 2 to generic functions of step 1.
4. Energy consumption measurement of the specific functions, to then aggregate those values into the generic functions.

8.3.1 Generic Decision Tree Breakdown

This is the first step in the proposed methodology where we identify the generic functions of a decision tree obtained from analyzing the *GrowTree* algorithm presented by Peter Flach [19]. Flach has identified four key functions: *homogeneous()*, *label()*, *bestSplit()* and *split()*. The *homogeneous()* function returns true if all the instances of the tree can be labeled with a single class. The *label()* function returns the label for the leaf node. There are different techniques to predict the value of the leaf node, e.g. using the majority class observed. The *bestSplit()* function returns the best attribute to split on. This can be achieved in different ways, such as using the *information gain* function. The *split()* function covers all the functions that are responsible for making the split of an internal node into different children.

8.3.2 Specific Function Breakdown

In the second step of the methodology we identify the specific functions of the VFDT. Specifically, we study the VFDT implementation from MOA (Massive Online Analysis) [20] version 2014.11. We have identified the structure of functions presented in Figure 8.1.

The training phase starts by calling the function `trainOnInstance()`, which reads each instance sequentially, and updates the tree by updating the statistics at the leaf after reading the instance. To do so, it calls `filterInstanceToLeaf()`, which sorts the instance to the leaf by following the tests at the nodes. Then, the function `learnFromInstance()` updates the statistics and labels the leaf based on the option set by the user (Majority class, Naive Bayes or a hybrid between both). Depending on the type of attribute (numerical or nominal) `newNominalClassObserver()` or

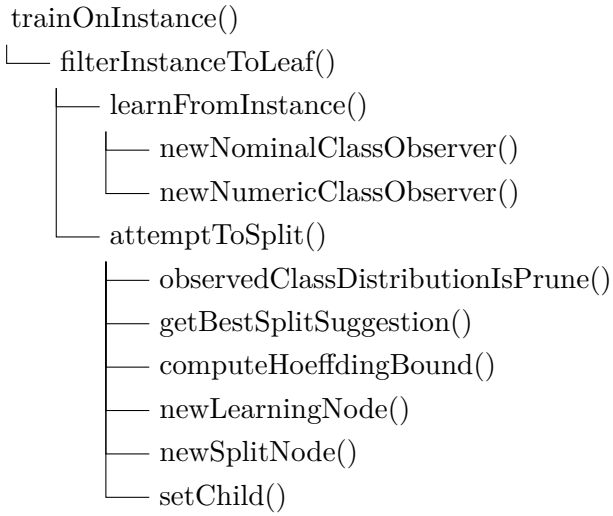

Figure 8.1: *Functions structure of the VFDT implementation.*

`newNumericClassObserver()` will be used to keep track of the attribute values at that leaf. `attemptToSplit()` will then decide between substituting the leaf with an internal node or keeping the leaf with the previously predicted class.

Inside `attemptToSplit()`, they first calculate if all instances observed so far belong to the same class with `observedClassDistributionIsPrune()`. If they do not belong to the same class, the Hoeffding Bound [4] is computed, by calling `computeHoeffdingBound()`. The two best attributes are obtained by calling the function `getBestSplitSuggestion()`. The difference between those attributes is compared with the Hoeffding Bound previously calculated. Thus, if such difference is higher than the Hoeffding Bound, there will be a split on the tree by replacing the leaf with a new internal node with the best attribute. This internal node is created by calling `newSplitNode()` and updated with the literals by creating new leaves calling `newLearningNode()` and `setChild()`. There are some functions specific to the MOA implementation that were not measured since we consider them a baseline: `estimateModelByteSizes()`, `calculateByteSizes()`, `findLearningNodes()`, `enforceTrackerLimit()`.

8.3.3 Specific To Generic Function Mapping

The third step, shown in Table 8.1, is to map each function of the VFDT to the functions of the generic decision tree. Most functions map to the *split()* and *label()* functions, since a significant part of the VFDT algorithm and implementation addresses different ways and functions on how to efficiently split the nodes. Both the *homogeneous()* and *bestSplit()* functions are used as in the generic decision tree.

Table 8.1: *Mapping of functions between the generic decision tree and the VFDT algorithm. First column=implementation functions (section 8.3.2). Second column=functions of the VFDT original algorithm [3]. Third column=generic functions of the decision tree (section 8.3.1).*

VFDT Implementation	VFDT algorithm	Decision Tree
`filterInstanceToLeaf()`	*Sort (x,y) into a leaf l ...*	*label()*
`learnFromInstance()`	*Label l and update statistics*	*label()*
`newNominalClassObserver()`	*Update statistics*	*label()*
`newNumericClassOsserver()`	*Update statistics*	*label()*
`observedClassDistributionIsPrune()`	*If the examples seen so far ...*	*homogeneous()*
`getBestSplitSuggestions()`	*Comp.* $\overline{G}_l(X_i)$*. Let* X_b *be the attribute...*	*bestSplit()*
`computeHoeffdingBound()`	*Compute* ϵ *using Equation 1*	*split()*
`newSplitNode()`	*Replace l by an internal node*	*split()*
`newLearningNode()`	*Add a new leaf* l_m *...*	*split()*
`setChild()`	*Add a new leaf* l_m *...*	*split()*

8.3.4 Energy Measurement

To estimate the energy consumption at the function level of the VFDT algorithm, we use the tool Jalen [9]. This tool accepts a Java or jar file as input, and outputs the energy consumption (in joules) of each function. Jalen outputs enough granularity to understand the energy consumed at the function level. The main limitation of the tool is the inability to estimate the energy consumption of programs that run during a short period of time, e.g. 5 seconds. We aggregate and combine the energy consumption of the specific VFDT functions to understand where the energy is being consumed in the generic decision tree functions.

8.4 Experimental Design

This experiment has been designed with two objectives:

- To understand which functions of the VFDT (mapped to generic decision tree functions) consume more energy than others.
- To understand the links between parameter configurations and the energy consumption, distributed across the generic functions. For instance, there could be some cases in which modifying the *tie threshold* parameter will affect the energy consumption of one specific function, and this function is the one that consumes more energy on average.

This knowledge makes it possible to make informed choices regarding parameter tuning to reduce energy consumption while retaining the same level of predictive accuracy.

8.4.1 Experimental Setup

The experiment is conducted as follows: The VFDT algorithm has been tuned with a total of 14 parameters setups (labeled A-N) and tested in four different datasets. Each configuration is an execution of the VFDT algorithm with such a parameter configuration. All 14 configurations have been tested on all four datasets. Therefore, there have been a total of $14 \times 4 = 56$ executions. Each execution has been repeated 5 times and averaged. The datasets and parameter tuning are further explained in Sections 8.4.2 and 8.4.3.

We evaluate the predictive performance and energy consumption of the VFDT under the different parameter setups and datasets. The training and testing of the algorithm are carried out in MOA (Massive Online Analysis) [20], and the energy is measured with Jalen (explained in Section 8.3.4). The experiment is run on a Linux machine with an i7@2.70 GHz and 8 GB of RAM.

8.4.2 Datasets

Our experiment features four different datasets, summarized in Table 8.2. The datasets have been synthetically generated from MOA. The Random Tree generator creates a tree, following the explanation from the VFDT original authors [3]. We consider this dataset as the default behavior of the algorithm. The Hyperplane generator uses a function to generate data that follows a plane in several dimensions [21]. This dataset is often used to test algorithms that can handle concept drift, making it a more challenging

synthetic dataset in comparison to the first one. The LED generator predicts the digit displayed on a LED display. Each attribute has a 10% chance of being inverted, and there a total of 7 segments in the display. Finally, the Waveform generator creates three different types of waves as a combination of two or three base waves. The goal is that the algorithm should be able to differentiate between these three types of waves [20].

Table 8.2: *Dataset summary.*

Dataset	Name	Type	Instances	Attributes	Numeric	Nominal
1	Random Tree	Synthetic	10,000,000	10	5	5
2	Hyperplane	Synthetic	10,000,000	10	10	-
3	LED	Synthetic	10,000,000	24	-	24
4	Waveform	Synthetic	10,000,000	21	21	-

8.4.3 Parameter Tuning

The parameters that have been varied are shown in Table 8.3. The *nmin* parameter represents the number of instances that the algorithm observes before calculating which attribute has the highest information gain. Theoretically, increasing this value will speed up the computations and lower the accuracy [3]. The τ parameter represents the tie threshold. Whenever the difference between the two best attributes is calculated, if this difference is smaller than τ, then there will be a split on the best attribute since the attributes are equally good. The absence of this parameter slows down the computation and decreases accuracy in theory [3]. The δ parameter represents one minus the confidence to make a split. In theory, the higher the confidence the higher the accuracy. The *Memory* parameter represents the maximum memory the tree can consume. When the memory limit is reached, the algorithm will deactivate less promising leaves. The last parameter is the leaf prediction parameter. This parameter was introduced in the VFDTc, an extension of the VFDT [22] that is able to handle numeric attributes and that features a Naive Bayes classifier to label the leaves. We test between majority class, Naive Bayes, or a hybrid between both (Naive Bayes Adaptive [23]). The hybrid calculates both the majority class and the Naive Bayes prediction, and chooses the one that outputs higher predictive performance. The goal is to discover if the extra computation done by calculating both Naive Bayes and the majority class in comparison

to calculating only one of them, trades-off with a significant increase in accuracy.

Table 8.3: *Parameter configuration index. Different configurations of the VFDT. The parameters that are changed are represented in bold.*

IDX	$nmin$	τ	δ	Memory	Leaf prediction
A	200	0.05	10^{-7}	30MB	NBA
B	**700**	0.05	10^{-7}	30MB	NBA
C	**1,200**	0.05	10^{-7}	30MB	NBA
D	**1,700**	0.05	10^{-7}	30MB	NBA
E	200	**0.01**	10^{-7}	30MB	NBA
F	200	**0.09**	10^{-7}	30MB	NBA
G	200	**0.13**	10^{-7}	30MB	NBA
H	200	0.05	$\mathbf{10^{-1}}$	30MB	NBA
I	200	0.05	$\mathbf{10^{-4}}$	30MB	NBA
J	200	0.05	$\mathbf{10^{-10}}$	30MB	NBA
K	200	0.05	10^{-7}	**100KB**	NBA
L	200	0.05	10^{-7}	**2GB**	NBA
M	200	0.05	10^{-7}	30MB	**NB**
N	200	0.05	10^{-7}	30MB	**MC**

NB = Naive Bayes. NBA = NB Adaptive. MC = Majority Class.

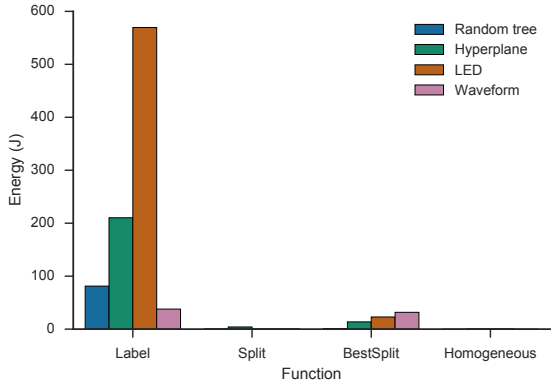

Figure 8.2: *Energy consumption, of the generic functions of a decision tree mapped from the VFDT algorithm, per dataset.*

8.5 Results and Analysis

The results of the experiment are shown in Tables 8.4 and 8.5. These values

Table 8.4: *Results for dataset 1 (random tree) and dataset 2 (hyperplane). In both datasets the energy is consumed in labeling (E1). The setup with lowest energy consumption and high accuracy is N: labeling with majority class. The random tree dataset outputs high accuracy in almost all setups.*

	Random tree dataset						Hyperplane dataset					
I	ACC	#Leaves	E1	E2	E3	ET	ACC	#Leaves	E1	E2	E3	ET
A	99.82	11,443	96.11	0.01	1.70	97.83	91.40	22,429	230.58	1.10	18.67	250.36
B	99.78	8,061	98.48	0.02	0.36	98.85	91.38	21,331	235.61	0.86	5.46	241.93
C	99.73	6,746	101.09	0.02	0.21	101.33	91.47	20,433	238.90	0.80	3.24	242.94
D	99.69	5,832	97.83	0.01	0.19	98.03	91.73	19,366	256.26	0.65	2.59	259.51
E	99.50	7,518	99.83	0.02	3.57	103.42	91.82	1,854	231.76	0.00	35.03	266.79
F	99.87	13,372	86.41	0.01	0.81	87.24	89.33	44,817	199.80	9.01	8.99	217.81
G	99.88	15,734	80.54	0.14	0.62	81.30	88.90	60,398	197.61	19.61	6.39	223.62
H	**99.90**	16,920	76.12	0.01	0.62	76.75	88.93	60,911	195.29	19.59	6.06	220.96
I	99.87	12,928	88.04	0.03	1.02	89.09	90.18	31,316	202.99	3.08	12.45	218.52
J	99.78	10,368	102.70	0.02	1.44	104.15	92.04	18,723	258.59	0.02	22.79	281.42
K	94.94	806	10.89	0.00	0.00	10.89	83.06	524	15.14	0.00	0.00	15.14
L	99.82	11,444	100.99	0.01	1.21	102.20	**92.29**	33,549	420.43	0.05	35.38	455.87
M	99.67	11,443	49.49	0.01	1.22	50.72	87.41	22,442	128.49	1.06	17.47	**147.04**
N	99.83	11,443	48.52	0.02	1.27	**49.81**	91.41	22,442	132.57	1.06	16.74	150.39
AVG	99.43	10,289	81.22	0.02	1.02	82.26	90.10	27,181	210.29	4.06	13.66	228.02

E1(J) = Energy in *label()*, E2(J) = Energy in *split()*, E3(J) = Energy in *bestSplit()*, E4(J) = Energy in *homogeneous()*, ET(J) = E1+E2+E3+E4. E4≃0 across all setups, thus it's omission.

are presented per parameter setup and in average. The values with higher accuracy and lower energy consumption are shown in bold. Setup K is not considered as the one with the lowest energy since its accuracy is significantly lower than the rest. Table 8.6 summarizes Tables 8.4 and 8.5 by averaging all setups. Each setup and dataset consume different amounts of energy. By comparing the parameters with the highest energy consumption and the lowest, we can have an energy saving of 52.18%, 47.76%, 74.22%, 76.24%, respectively per dataset.

Figure 8.4 shows the trade-off between accuracy and energy. The correlation between these two variables is of -0.79, which suggests that the higher the accuracy the lower the energy consumption, calculated from Table 8.6. The reason for this correlation is that datasets such as the LED dataset are complicated to analyze, thus consuming a lot of energy and obtaining low accuracy results. The opposite occurs in the Random Tree, that is easy to analyze, thus consuming less energy to do so, outputting higher accuracy results.

Table 8.5: *Results for dataset 3 (LED) and dataset 4 (waveform). In the LED dataset, accuracy is constant and the energy consumption very high, most of it consumed in labeling when there are less splits on the nodes. In the Waveform dataset energy is consumed between labeling (E1) and calculating the best split (E3).*

	LED dataset						Waveform dataset					
I	ACC	#Leaves	E1	E2	E3	ET	ACC	#Leaves	E1	E2	E3	ET
A	74.02	2,789	835.91	0.01	35.38	871.31	84.98	11,852	30.98	0.02	41.13	72.14
B	74.02	2,788	836.07	0.01	10.09	846.17	85.04	11,262	29.35	0.00	12.09	41.44
C	74.01	2,772	832.26	0.01	6.20	838.47	85.06	10,794	29.38	0.00	6.85	36.23
D	74.02	2,780	833.92	0.01	4.37	838.31	85.04	10,377	27.70	0.01	5.02	**32.73**
E	**74.03**	176	490.50	0.00	45.32	535.81	85.42	868	6.77	0.01	90.33	97.11
F	74.00	7,437	549.87	0.01	19.71	569.59	84.45	24,557	58.23	0.00	21.57	79.81
G	74.02	13,337	388.73	0.01	12.00	400.75	84.25	36,935	81.10	0.00	13.36	94.46
H	74.00	13,498	384.36	0.02	11.58	395.97	84.24	37,839	79.86	0.00	13.20	93.05
I	74.00	4,466	720.77	0.01	27.44	748.23	84.74	17,027	43.52	0.01	31.08	74.62
J	74.02	2,331	831.28	0.00	37.33	868.60	85.24	9,035	25.08	0.01	52.41	77.50
K	74.02	285	17.73	0.00	0.06	17.79	81.12	438	0.39	0.00	0.08	0.48
L	74.02	2,789	860.54	0.01	36.19	896.75	**85.59**	19,029	58.56	0.04	79.16	137.75
M	74.01	2,789	186.68	0.00	37.97	**224.65**	83.58	11,866	28.83	0.01	39.04	67.89
N	74.01	2,789	204.14	0.00	37.40	241.54	84.95	11,866	30.10	0.23	38.88	69.21
AVG	74.01	4,359	569.48	0.01	22.93	592.42	84.55	15,267	37.85	0.02	31.73	69.60

E1(J) = Energy in *label()*, E2(J) = Energy in *split()*, E3(J) = Energy in *bestSplit()*, E4(J) = Energy in *homogeneous()*, ET(J) = E1+E2+E3+E4. E4≃0 across all setups, thus it's omission.

8.5.1 Energy Hotspot

The aim of this paper is to profile the energy consumption of the VFDT by showing which part of the algorithm is consuming most of the energy. Figure 8.3 shows the functions that consume the most amount of energy in the algorithm. These functions, mapped to generic functions, are shown in Figure 8.2. The energy hotspot of the VFDT is `learnFromInstance()`, that maps to the labeling phase of the algorithm. This function has two objectives.

Table 8.6: *Averaged accuracy, leaves and energy results, from executing the VFDT algorithm in the four datasets shown in Table 8.2. Averaged from Tables 8.4 and 8.5.*

Dataset	ACC(%)	#Leaves	E1(J)	E2(J)	E3(J)	E4(J)	ET(J)
Random Tree	99.43	10,289	81.22	0.02	1.02	0.00	82.26
Hyperplane	90.10	27,181	210.29	4.06	13.66	0.01	228.02
LED	74.01	4,359	569.48	0.01	22.93	0.00	592.42
Waveform	84.55	15,267	37.85	0.02	31.73	0.00	69.60

E1 = Energy in *label()*, E2 = Energy in *split()*, E3 = Energy in *bestSplit()*, E4 = Energy in *homogeneous()*, ET = E1+E2+E3+E4.

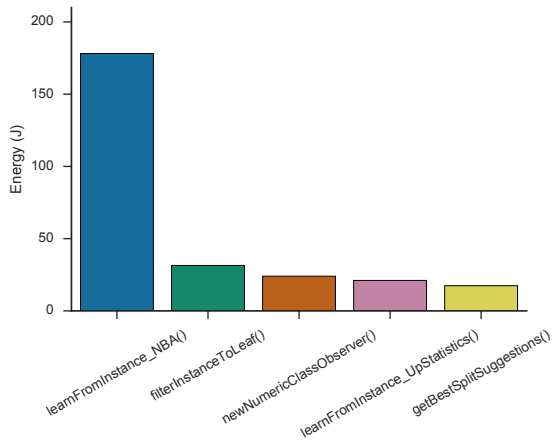

Figure 8.3: *Five functions with highest energy consumption.*

The first one, represented by `learnFromInstance_UpStatistics()` in the mentioned figure, updates the statistics of every leaf. The second one, represented by `learnFromInstance_NBA()`, predicts the class at the leaf on every iteration of the algorithm applying Naive Bayes and the majority class, to keep count of which one is giving a better predictive performance. The second function is only active when the parameter NBA (Naive Bayes Adaptive) is set (by default in the implementation). Based on the results, it is more energy efficient to split on a node rather than to delay the splitting, because updating the statistics at a leaf and applying Naive Bayes is more expensive on leaves that have already seen many examples. This phenomena can be observed in datasets 1,2 and 3.

Dataset 4 has a different behavior than datasets 1-3 because it contains only numeric attributes whose values are not close to each other. The energy consumed in the labeling phase of this dataset is mainly done by the function `newNumericClassObserver()`, because updating the statistics of such attributes is quite expensive. However, this function consumes less energy when there are less splits on the leaves, as opposed to the behavior of `learnFromInstance()`. That is why in dataset 4, more splits lead to a higher energy consumption.

In summary, updating the statistics and setting the leaf prediction to Naive Bayes Adaptive are the energy hotspots of the algorithm. They are very expensive operations, and their energy consumption is higher when the

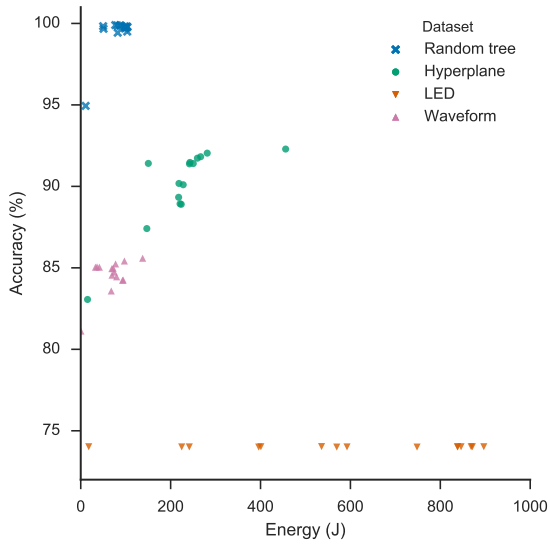

Figure 8.4: *Trade-off between accuracy and energy for all setups and datasets. Each dataset has a different pattern of energy consumption. The random tree has the highest accuracy and lowest energy consumption.*

splits on the leaves are delayed. There is a trade-off regarding the numeric estimator, since it consumes less energy whenever the splits are delayed. So, if the data has numeric attributes that are complicated to keep track of, it might be better to have smaller trees, since this can reduce the energy of the numeric estimator, and this function might be consuming more energy than the others in `learnFromInstance()`.

8.5.2 Parameter Analysis

We focus on the analysis of such parameters that can reduce the energy consumption of the labeling phase, and in particular of `learnFromInstance()`, since that is the energy hotspot. The longer it takes to make a decision to split on a leaf (by substituting it with a node), the higher the energy consumption. τ and δ are two parameters that have a great impact on how the tree expands. $1 - \delta$ represents the confidence on making the correct split. The higher the δ, the lower the confidence, leading to more splits and a reduction in energy. τ represents how separate the attributes can be in order to split. The higher the τ, the easier it is to make a split, thus lowering

the energy consumption. For this reason, $\tau = 0.13$ and $\delta = 10^{-1}$ lead to very similar energy patterns. This occurs in datasets 1-3, for the reasons explained in the previous subsection.

In relation to *leaf prediction*, the default setup for this parameter (*Naive Bayes Adaptive*) is, on average, a worse choice than the other possible setups: *Naive Bayes* or *majority class*. For all datasets, the choice of one of these two setups showed a reduction in energy, hardly affecting accuracy. NBA computes the leaf prediction by applying Naive Bayes and majority class on every iteration of the algorithm for each instance, thus being very computationally and energy inefficient. On the other hand, *Naive Bayes* and *Majority class* predict the leaf value on the testing phase, reducing the energy consumption significantly. Moving on to *memory consumption*, we observe that reducing the amount of memory allocated for the tree drastically decreases energy consumption, but it also significantly reduces accuracy. We propose for future work to tune different values of memory consumption to see if there are important savings without having to reduce the accuracy of the model. Finally, when increasing the *nmin* parameter the accuracy and energy consumption are not significantly affected.

8.5.3 Suggestions to Improve Energy Efficiency

Based on this analysis, we can extract some suggestions to significantly reduce the energy consumption of the VFDT.

- Avoiding using the parameter Naive Bayes Adaptive since it marginally increases accuracy and it consumes a lot of energy. The reason is that Naive Bayes and majority class are computed every time an instance is processed.

- Splitting the leaf into a node is more energy efficient than delaying the split, since updating the statistics at the leaves that have already observed many instances is very expensive.

- Increasing the δ and τ has positive effects on energy consumption, creating more splits and avoiding delaying the splits.

- If the data is numerical and complicated to keep track of, delaying the splits can be more energy efficient.

8.6 Conclusions and Future Work

The aim of this paper is to profile the energy consumption of the Very Fast Decision Tree algorithm. To achieve that, we have presented the following: (i) a methodology that profiles the energy consumption of decision trees at the function level, (ii) an experiment that uses this methodology to discover the most energy consuming functions of the VFDT, (iii) a thorough analysis to understand the reasons behind the functions energy consumption, and (iv) which parameter settings decrease the energy consumption of such functions.

The analysis of the results show that the functions responsible for the labeling of the decision tree are the main energy hotspots. Specifically, updating the statistics in the leaves and predicting the class at the leaf with parameter NBA are the main reasons for the amount of energy consumed in the algorithm. The energy consumption in labeling is significantly reduced if there are more splits on the leaves, rather than delaying the splitting to gain a higher confidence. Additionally, the results show that the energy can be reduced up to a 74.3% by tuning the parameters of the VFDT.

The planned future work is to make an energy efficient extension of the VFDT with the knowledge extracted from this study. We will also investigate in more depth different parameter combinations to see if energy can be reduced further. We also plan to compare the predictive performance and the energy consumption of other tree learners with the VFDT.

8.7 References

[1] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding. “Data mining with big data”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.1 (2014), pp. 97–107.

[2] G. De Francisci Morales. “SAMOA: A platform for mining big data streams”. In: *Proceedings of the 22nd International Conference on World Wide Web*. ACM. 2013, pp. 777–778.

[3] P. Domingos and G. Hulten. “Mining high-speed data streams”. In: *Proc. 6th SIGKDD Int’l Conf. on Knowledge discovery and data mining*. 2000, pp. 71–80.

[4] W. Hoeffding. "Probability inequalities for sums of bounded random variables". In: *Journal of the American statistical association* 58.301 (1963), pp. 13–30.

[5] C. Reams. "Modelling energy efficiency for computation". PhD thesis. University of Cambridge, 2012.

[6] S. Murugesan. "Harnessing green IT: Principles and practices". In: *IT professional* 10.1 (2008), pp. 24–33.

[7] A. Hooper. "Green computing". In: *Communication of the ACM* 51.10 (2008), pp. 11–13.

[8] A. Freire, C. Macdonald, N. Tonellotto, I. Ounis, and F. Cacheda. "A self-adapting latency/power tradeoff model for replicated search engines". In: *7th ACM international conference on Web search and data mining*. 2014, pp. 13–22.

[9] A. Noureddine, R. Rouvoy, and L. Seinturier. "Monitoring energy hotspots in software". In: *Automated Software Engineering* 22.3 (2015), pp. 291–332.

[10] R. D. King, C. Feng, and A. Sutherland. "Statlog: comparison of classification algorithms on large real-world problems". In: *Applied Artificial Intelligence an International Journal* 9.3 (1995), pp. 289–333.

[11] R. Caruana and A. Niculescu-Mizil. "An empirical comparison of supervised learning algorithms". In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 161–168.

[12] N. K. Ahmed, A. F. Atiya, N. E. Gayar, and H. El-Shishiny. "An empirical comparison of machine learning models for time series forecasting". In: *Econometric Reviews* 29.5-6 (2010), pp. 594–621.

[13] A. Murdopo. "Distributed decision tree learning for mining big data streams". In: (2013).

[14] N. Kourtellis, G. D. F. Morales, A. Bifet, and A. Murdopo. "VHT: Vertical Hoeffding Tree". In: *arXiv preprint arXiv:1607.08325* (2016).

[15] M. Demirci. "A Survey of Machine Learning Applications for Energy-Efficient Resource Management in Cloud Computing Environments". In: *IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. 2015, pp. 1185–1190.

[16] T.-J. Yang, Y.-H. Chen, and V. Sze. “Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning”. In: *arXiv preprint arXiv:1611.05128* (2016).

[17] E. García Martín, N. Lavesson, and H. Grahn. “Energy Efficiency in Data Stream Mining”. In: *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*. ACM. 2015, pp. 1125–1132.

[18] E. García-Martín, N. Lavesson, and H. Grahn. “Energy Efficiency Analysis of the Very Fast Decision Tree Algorithm”. In: *Trends in Social Network Analysis - Information Propagation, User Behavior Modelling, Forecasting, and Vulnerability Assessment. (To appear in April 2017)*. Ed. by R. Missaoui, T. Abdessalem, and M. Latapy. 2016.

[19] P. Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

[20] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. “MOA: Massive online analysis”. In: *The Journal of Machine Learning Research* 11 (2010), pp. 1601–1604.

[21] G. Hulten, L. Spencer, and P. Domingos. “Mining time-changing data streams”. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 97–106.

[22] J. Gama, R. Rocha, and P. Medas. “Accurate decision trees for mining high-speed data streams”. In: *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2003, pp. 523–528.

[23] R. B. Kirkby. “Improving hoeffding trees”. PhD thesis. The University of Waikato, 2007.

9

Energy-Aware Very Fast Decision Tree

Eva García-Martín, Niklas Lavesson, Håkan Grahn, Emiliano Casalicchio, Veselka Boeva

Abstract

Recently machine learning researchers are designing algorithms that can run in embedded and mobile devices, which introduces additional constraints compared to traditional algorithm design approaches. One of these constraints is energy consumption, which directly translates to battery capacity for these devices. Streaming algorithms, such as the Very Fast Decision Tree (VFDT), are designed to run in such devices due to their high velocity and low memory requirements. However, they have not been designed with an energy efficiency focus. This paper addresses this challenge by presenting the *nmin adaptation* method, which reduces the energy consumption of the VFDT algorithm with only minor effects on accuracy. *nmin adaptation* allows the algorithm to grow faster in those branches where there is more confidence to create a split, and delays the split on the less confident branches. This removes unnecessary computations related to checking for splits but maintains similar levels of accuracy. We have conducted extensive experiments on 29 public datasets, showing that the VFDT with *nmin adaptation* consumes up to 31% less energy than the original VFDT, and up to 96% less energy than the CVFDT (VFDT adapted for concept drift scenarios), trading off up to 1.7 percent of accuracy.

9.1 Introduction

State-of-the-art machine learning algorithms are now being designed to run in the edge, which creates new time, memory, and energy requirements. Streaming algorithms fulfill the time and memory requirements by building models in real-time, processing data with high velocity, and low memory consumption. The Very Fast Decision Tree (VFDT) algorithm [1] was

the first streaming algorithm with the aforementioned properties, that still achieves competitive accuracy results. However, energy consumption has not been considered during the design of the VFDT and state-of-the-art streaming algorithms. Since machine learning algorithms are starting to be designed in the edge [2, 3], we believe that it is not feasible anymore to build algorithms that are not energy-aware. To address this gap, this paper presents the *nmin adaptation* method, which reduces the energy consumption of the VFDT and other Hoeffding tree algorithms, with only minor effects on accuracy.

The *nmin adaptation* method adapts the value of the *nmin* parameter on real-time and based on the incoming data. The *nmin* parameter sets the minimum number of observed instances (batch size) at each leaf to check for a possible split. By setting a unique and adaptive value of *nmin* at each leaf, this method allows the tree to grow faster on those paths where there are clear splits, while delaying splits on more uncertain paths. By delaying the growth in those paths where there is not enough confidence, the algorithm saves significant amount on energy on unnecessary tasks, with only minor effects on accuracy.

This paper extends our previous work "Hoeffding Trees with *nmin adaptation*" [4] by adding extensive experiments that validate the proposed method with statistical tests on the results. The results clearly show how the VFDT with *nmin adaptation* (entitled VFDT-*nmin*) obtains significantly lower levels of energy consumption (7 percent on average, up to a 31 percent) affecting accuracy up to a 1.7%. In particular, we have investigated the energy consumption and accuracy of the VFDT, VFDT-*nmin*, and CVFDT (Concept-Adapting Very Fast Decision Tree [5]) algorithms under 29 public datasets.

We first investigated the energy consumption and accuracy of the mentioned algorithms in a baseline scenario, where we empirically validated that handling numerical attributes is much more energy consuming than handling nominal attributes. We then examined the effect of concept drift on the mentioned algorithms, concluding that i)VFDT-*nmin* achieves significantly lower energy consumption than VFDT and CVFDT on the majority of datasets, ii) VFDT-*nmin* scales better than VFDT (and CVFDT) in terms of energy consumption when increasing the amount of drift, and iii) CVFDT (designed to handle concept drift) obtains lower levels of accuracy in the majority of concept drift datasets, while VFDT and VFDT-*nmin* perform

similarly. Finally, we showed how VFDT-*nmin* obtains significantly lower levels of energy consumption in 5/6 real world datasets, while obtaining the highest accuracy in all real world datasets.

The contributions of this paper are summarized as follows:

- We present the *nmin adaptation* method, to create energy-aware Hoeffding tree algorithms, which makes them suitable for running in the edge.
- We present a general approach to create energy models for different classes of machine learning algorithms. We apply this knowledge to create an energy model for the VFDT, independent of the programming language and hardware platform. One of the findings of the model is the high energy consumption of handling numerical attributes compared to nominal attributes.
- We empirically validate the previous claim in Section 9.6.2, where we show how handling numerical attributes consumes up to 12X more energy than nominal attributes. This is visible also in Fig. 9.6.
- We show how VFDT-*nmin* scales better than the VFDT in terms of energy consumption when increasing the amount of drift.
- We show how VFDT-*nmin* consumes significantly less energy than the VFDT on 76% of the datasets (7% on average, up to a 31%), and than the CVFDT on all the datasets, (86% on average, up to a 97%), while affecting accuracy by less than 1%, up to a 1.7%. These claims are validated with statistical tests on the data.

In this extension we expanded the datasets from 15 to 29, investigating more phenomena such as concept drift, and validating its use in real world datasets. We also performed statistical tests on the results, validating that there is no significant difference between the accuracy of the VFDT and the VFDT-*nmin*, and that the VFDT-*nmin* consumes significantly less energy than the VFDT. We also added Section 4.3, that explains a general way to create energy models for different classes of algorithms. Finally, we theoretically bounded the batch size of instances that VFDT-*nmin* can adapt to, showing that it can affect accuracy by at most $\frac{\delta}{p}$, where δ is the confidence value and p the leaf probability.

The rest of the paper is organized as follows. The background and related work are presented in Section 9.2. The *nmin adaptation* method is presented in Section 9.3. The energy model that profiles the energy consumption of the VFDT is presented in Section 9.4. Section 9.5 presents the experimental design. Sections 9.6 present the results and discussion. Section 9.7 details the limitations of this study. Section 9.8 concludes the paper with the significance and impact of our work.

9.2 Background and Related Work

In this section we explain the fundamentals of the VFDT. In addition, we introduce related studies in streaming data, and resource aware machine learning.

9.2.1 VFDT

Very Fast Decision Tree [1] is a decision tree algorithm that builds a tree incrementally. The data instances are analyzed sequentially and only once. The algorithm reads an instance, sorts it into the corresponding leaf, and updates the statistics at that leaf. To update the statistics the algorithm maintains a table for each node, with the observed attribute and class values. Updating the statistics of numerical attributes is done by saving and updating the mean and standard deviation for every new instance. Each leaf also stores the instances observed so far. After *nmin* instances are read at that leaf, the algorithm calculates the information gain ($\overline{G}$) from all observed attributes. The difference in information gain between the best and the second best attribute ($\Delta\overline{G}$) is compared with the Hoeffding Bound [6] (ϵ). If $\Delta\overline{G} > \epsilon$, then that leaf is substituted by a node, and there is a split on the best attribute. That attribute is removed from the list of attributes available to split in that branch. If $\Delta\overline{G} < \epsilon < \tau$, a tie occurs, splitting on any of the two top attributes, since they have very similar information gain values. The Hoeffding Bound (ϵ),

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{9.1}$$

states that the chosen attribute at a specific node after seeing n number of examples, will be the same attribute as if the algorithm has seen an infinite number of examples, with probability $1 - \delta$.

We now discuss the computational complexity of the VFDT, shown in lines 1-21 from Alg. 6. Suppose that n is the number of instances and m is the number of attributes. The algorithm loops over n iterations. Every step between 6 and 9 requires execution time that is proportional to m. In the worst case scenario the computational complexity of step 7 is $O(m)$ according to [1]. The function in step 7 traverses the tree until it finds the corresponding leaf. Since the attributes are not repeated for each branch, in the worst case scenario the tree will have a depth of m attributes. Step 8 runs in constant time. The computational complexity of this part can be evaluated to $O(n \cdot m)$. The computational complexity of the remainder part of the algorithm (from step 11 downwards) depends on $n/nmin$. Moreover, the computational complexity of steps 11 to 13 is equal to $O(m)$, while steps 16 to 18 need constant time, i.e. the computational complexity of this part is $O(n/nmin \cdot m)$. The total computational complexity of the VFDT is $O(n \cdot m) + O(n/nmin \cdot m)$ and $n >> nmin$, i.e. it can be simplified to $O(n \cdot m)$.

9.2.2 Related Work

Energy efficiency is an important research topic in computer engineering [7]. Reams et al. [8] provide a good overview of energy-efficiency in computing for different platforms: servers, desktops, and mobile devices. The authors also proposes an energy cost model based on the number of instructions, power consumption, the price per unit of energy, and the execution time. While energy efficiency has mostly been studied in computer engineering, during the past years green computing has emerged. Green IT, also known as green computing, started in 1992 with the launch of the Energy Star program by the US Environmental Protection Agency (EPA) [9]. Green computing is *the study and practice of designing, manufacturing, using, and disposing computers, servers, and associated systems efficiently and effective with minimal or no environmental impact* [9]. One specific area is energy-efficient computing [8], where there is a significant focus on reducing the energy consumption of data centers [10].

In relation to big data, data centers, and cloud computing, there have been several studies that design methods for energy-efficient cloud computing [11, 12]. One approach was used by Google Deep Mind to reduce the energy used in cooling their data centers [13]. These studies focused on reducing the energy consumed by data centers using machine learning to,

e.g., predict the load for optimization. However, we focus on reducing the energy consumption of machine learning algorithms.

Regarding machine learning and energy efficiency, there has been a recent increase of interest towards resource-aware machine learning. The focus has been on building energy efficient algorithms that are able to run on platforms with scarce resources [14–17]. Closely related is the work done on building energy-efficient deep neural networks [2, 18]. They developed a model where the energy cost of the principal components of a neural network is defined, and then used for pruning a neural network without reducing accuracy. Some work is also conducted in building energy and computational efficient cluster solutions [19], accelerating the original algorithm by parallelizing some of the key features.

Data stream mining algorithms analyze data aiming at reducing the memory usage, by reading the data only once without storing it. Examples of efficient algorithms are the VFDT [1] and a KNN streaming version with self-adjusting memory [20]. There have been extensions to these algorithms for distributed systems, such as the Vertical Hoeffding Tree [21], where the authors parallelize the induction of Hoeffding Trees; and the Streaming Parallel Decision Tree algorithm (SPDT). Applications of streaming algorithms have been conducted in many domains, such as fraud detection [22] and time series forecasting [23]. More focused on hardware approaches to improve Hoeffding trees is the work proposed by [24], where they parallelize the execution of random forest of Hoeffding trees, together with a specific hardware configuration to improve induction of Hoeffding trees. Other work has been done where the authors present the energy hotspots of the VFDT [25]. Our proposed work in this paper focuses on a direct approach to reduce the energy consumption of the VFDT by dynamically adapting the *nmin* parameter based on incoming data, introducing the notion of *dynamic parameter adaptation* in data stream mining.

9.3 *nmin* adaptation

The *nmin adaptation* method, the main contribution of this paper, aims at reducing the energy consumption of the VFDT while maintaining similar levels of accuracy. There are many computations and memory accesses dependent on the parameter *nmin*, observed in the energy model presented in Section 9.4. However, the design of the original VFDT sets the value

of *nmin* to a fixed value from the beginning of the execution. This is problematic, because there are many functions that would be computed unnecessarily if the number of *nmin* instances is not high enough to make a confident split (e.g., *calc_entropy*, *calc_hoeff_bound*, and *get_best_att*). Our goal is to set *nmin* to a specific value on each leaf that ensures a split, so that the $\frac{N}{nmin}$ values in Eq. 9.20 are only computed when needed. *nmin adaptation* adapts the value of *nmin* to a higher one, thus making $\frac{N}{nmin}$ smaller. This approach reduces computations, reduces memory accesses, and doesn't affect the final accuracy, since we are only computing those functions when needed.

In another publication, the authors [25] already confirmed the high energy impact of the functions involved in calculating the best attributes. This matches with our energy model, and motivates the reasons and objectives for *nmin adaptation*:

1. Reduce the number of computations and memory accesses by adapting the value of *nmin* to a specific value on each leaf that ensures a split.

2. Maintain similar levels of accuracy by removing only unnecessary computations, thus developing the same tree structure.

nmin adaptation sets *nmin* to the estimated number of instances required to guarantee a split with confidence $1 - \delta$. The higher the value of *nmin*, the higher the chance to split. However, setting *nmin* to a very high value can decrease accuracy if the growth of the tree is significantly delayed, and setting *nmin* to a lower value increases the accuracy at the expense of energy, as it has to calculate the $\overline{G}$ of all attributes even when there are not enough instances to make a confident split. Thus, our solution allows for a faster growth in the branches with a higher confidence to make a split, and delays the growth in the less confident ones. We have identified two scenarios that are responsible for not splitting. We set *nmin* to a different value to address these scenarios, depending on the incoming data. The two scenarios are the following:

Scenario 1 ($\Delta\overline{G} < \epsilon$ and $\Delta\overline{G} > \tau$)**.**
Fig. 9.1, left plot. The attributes are not too similar, since $\Delta\overline{G} > \tau$, but their difference is not big enough to make a split, since $\Delta\overline{G} < \epsilon$. The solution is to wait for more examples until ϵ (green triangle) decreases and is smaller than

$\Delta\overline{G}$ (black star). Following this reasoning, $nmin = \left\lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot (\Delta G)^2} \right\rceil$, obtained by setting $\epsilon = \Delta\overline{G}$ in (9.1), to guarantee that $\Delta\overline{G} \geq \epsilon$ will be satisfied in the next iteration, creating a split.

Scenario 2 ($\Delta\overline{G} < \epsilon$ and $\Delta\overline{G} < \tau$ but $\epsilon > \tau$)**.**
The top attributes are very similar in terms of information gain, but ϵ is still higher than τ, as can be seen in Fig. 9.1, right plot. The algorithm needs more instances so that ϵ (green triangle) decreases and is smaller than τ (red dot). Following this reasoning, $nmin = \left\lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot \tau^2} \right\rceil$, by setting $\epsilon = \tau$ in (9.1). In the next iteration $\epsilon \leq \tau$ will be satisfied, forcing a split.

The upper bound of the batch size is given in scenario 2, since the lower the ϵ the higher the number of instances. The lowest value of ϵ is when $\epsilon = \tau$, because if $\epsilon < \tau$ then a split occurs, so there would be no need to adapt the value of *nmin* in that case. The lower bound of the batch size is given in scenario 1, and for the case when $\Delta G = \epsilon$, which is approximately the initial value of *nmin*. Thus, the adaptive size of the batch can be bounded to the following interval: $\left\lceil \text{initial } nmin, \frac{R^2 \cdot ln(1/\delta)}{2 \cdot \tau^2} \right\rceil$.

The upper and lower bound can be related to the notion of *intensional disagreement*, which is described in the VFDT original paper as *the probability that the path of an example through DT1 will differ from its path through DT2* [1]. There, the authors propose *Theorem 1*, which states that the intensional disagreement between the Hoeffing tree and the batch tree is lower than $\frac{\delta}{p}$, where p is the leaf probability.

The original VFDT and the VFDT-*nmin* only differ in the way the node split is created. The VFDT-*nmin* requires that more examples are seen at a node to make an informed decision. Based on the VFDT original paper, the authors confirm that when more examples are read at the node, i.e. increasing *nmin*, the value of δ decreases, increasing the confidence of the split. In this case, increasing the *nmin* would create a tree that at most differs with the original (batch) tree in $\frac{\delta}{p}$, in a scenario with an infinite data stream. Since in our case the upper bound (around 3,000 instances) is significantly lower than the size of the stream (around 1 million instances), accuracy should not be significantly affected.

The pseudocode of VFDT-*nmin* is presented in Alg. 6. The specific part

of *nmin adaptation* is shown in lines 22-26, where we specify how *nmin* is going to be adapted based on the scenarios explained above. The idea is that, when those scenarios occur, we adapt the value of *nmin*, so that they don't occur in the next iteration, thus ensuring a split. Fig. 9.2 shows a diagram of the main functions and functionalities of the VFDT-*nmin* algorithm, inspired in the work by [26].

In relation to the computational complexity of the *nmin adaptation*, we can observe that this method does not add any overhead. Thus, the computational complexity of VFDT-*nmin* is $O(n \cdot m)$.

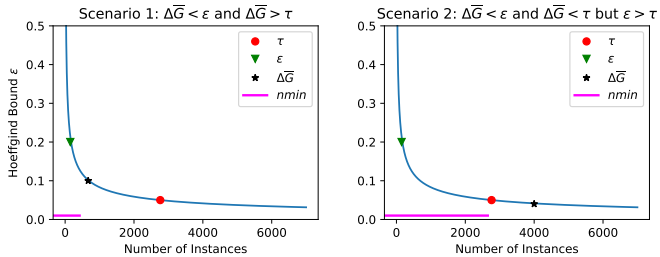

Figure 9.1: *Variation of ϵ (Hoeffding bound) with the number of instances. nmin adaptation method for scenarios 1 and 2.*

In relation to concept drift, state-of-the-art Hoeffding tree algorithms that are able to handle concept drift [27] also use the *nmin* parameter to decide when to check for possible splits. Thus, our method can be directly apply to those class of algorithms and should theoretically output similar results. We have planned that for future works.

Finally, we show an example of how *nmin adaptation* works for two of the datasets used in the final experiments. These datasets are described in Table 9.1. Fig. 9.3 shows the *nmin* variation for the cases when *nmin* is initially set to $20, 200, 2000$. So, after those instances, *nmin* will adapt to a higher value depending on the data observed so far at that specific leaf. The airline dataset shows many adaptations of *nmin* when *nmin* is initially set to 20. This is expected, since we are showing the adaptations per leaf, so at the beginning all the leaves starting with $nmin = 20$ will adapt that value to a much larger one. The same reasoning occurs when $nmin = 200$ initially, since there will be less adaptations because the leaves need to wait for more instances, and there is a higher chance to split when more instances are read. The poker dataset exhibits a different behavior, where *nmin* adapts to a

Algorithm 6 VFDT-*nmin*: Very Fast Decision Tree with *nmin adaptation*

HT: Tree with a single leaf (the root)
X: set of attributes
$G(\cdot)$: split evaluation function
τ: hyperparameter set by the user
$nmin$: hyperparameter initially set by the user
while stream is not empty **do**
  Read instance I_i
  Sort I_i to corresponding leaf l using HT
  Update statistics at leaf l
  Increment n_l: instances seen at l
  if $nmin \leq n_l$ **then**
    Compute $\overline{G_l}(X_i)$ for each attribute X_i
    X_a, X_b = attributes with the highest $\overline{G_l}$
    $\Delta\overline{G} = \overline{G}_l(X_a) - \overline{G}_l(X_b)$
    Compute ϵ
    if $(\Delta\overline{G} > \epsilon)$ or $(\epsilon < \tau)$ **then**
      Replace l with a node that splits on X_a
      for each branch of the split **do**
        New leaf l_m with initialized statistics
      end for
    else
      Disable attr $\{X_p|(\overline{G}_l(X_p) - \overline{G}_l(X_a)) > \epsilon\}$
      if $\Delta\overline{G} \leq \tau$ **then**

        $nmin = \left\lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot \tau^2} \right\rceil$
      else
        $nmin = \left\lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot (\Delta G)^2} \right\rceil$
      end if
    end if
  end if
end while

higher value, 30491. This occurs in Scenario 2, but since the poker dataset has 10 classes, the range R of the Hoeffding bound equation (9.1) is higher. Finally, looking at the cases where $nmin = 2000$ (green), we observe how there is almost no adaptation. VFDT-*nmin* either splits after 2000 instances,

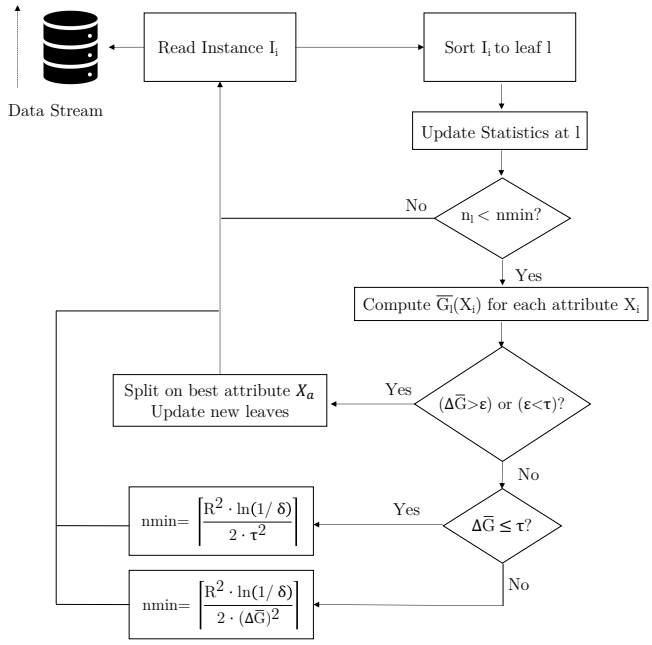

Figure 9.2: *Flowchart diagram of the VFDT-nmin algorithm.*

or it adapts $nmin = 2,763$ or $nmin = 30,491$, because the attributes are very similar.

9.4 Energy Consumption of the VFDT

Energy consumption is a necessary measurement for today's computations, since it has a direct impact on the electricity bill of data centers, and battery life of embedded devices. However, measuring energy consumption is a challenging task. As has been shown by researchers in computer architecture, estimating the energy consumption of a program is not straightforward, and is not as simple as measuring the execution time, since there are many other variables involved [28].

In this section, we first give a general background on energy consumption and its relationship to software energy consumption. We then propose a general approach to create energy models applicable to any class of algorithms. We then use this approach to create a theoretical energy model for the VFDT

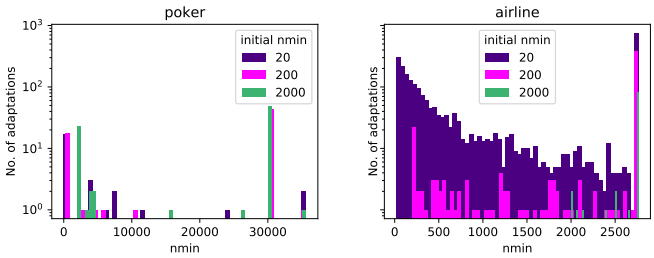

Figure 9.3: *Variation of nmin for nmin initially set to* $20, 200, 2000$ *on poker and airline datasets (Table 9.1). With a lower nmin, nmin adaptation adapts nmin to a higher value more frequently. The peaks on* $nmin = 2,763$ *and* $nmin = 30,491$ *is explained by Scenario 2, since* τ *is a fixed hyperparameter.*

algorithm, based on the number of instances of the stream, and number of numerical and nominal attributes.

9.4.1 General energy consumption

Energy efficiency in computing usually refers to a hardware approach to reduce the power consumption of processors, or ways to make processors handle more operations using the same amount of power [29].

Power is the rate at which energy is being consumed. The average power during a time interval T is defined as [30]:

$$P_{avg} = \frac{E}{T} \tag{9.2}$$

where E, energy, is measured in joules (J), P_{avg} is measured in watts (W), and time T is measured in seconds (s). We can distinguish between dynamic and static power. Static power, also known as leakage power, is the power consumed when there is no circuit activity. Dynamic power, on the other hand, is the power dissipated by the circuit, from charging and discharging the capacitor [7]:

$$P_{dynamic} = \alpha \cdot C \cdot V_{dd}^2 \cdot f \tag{9.3}$$

where α is the activity factor, representing the percentage of the circuit that is active. V_{dd} is the voltage, C the capacitance, and f the clock frequency measured in hertz (Hz). Energy is the effort to perform a task, and it is

defined as the integral of power over a period of time [7]:

$$E = \int_0^T P(t)dt \tag{9.4}$$

In this study we focus on the measurement of energy consumption, since it gives an overview of how much power is consumed in an interval of time.

Finally, we conclude with an explanation of how programs consume energy. The total execution time of a program is defined as [7]:

$$T_{exe} = IC \times CPI \times T_c \tag{9.5}$$

where IC is the number of executed instructions, CPI (clock cycles per instruction) is the average number of clock cycles needed to execute each instruction, and T_C is the clock cycle time of the processor. The total energy consumed by a program is:

$$E = IC \times CPI \times EPC \tag{9.6}$$

where EPC is the energy per clock cycle, and it is defined as

$$EPC \propto C \cdot V_{dd}^2 \tag{9.7}$$

The value CPI depends on the type of instruction, since different instructions require different number of clock cycles to complete. However, measuring only time does not give a realistic view on the energy consumption, because there are instructions that can consume more energy due to a long delay (e.g., memory accesses), or others that consume more energy because of a high requirement of computations (floating point operations). Both could obtain similar energy consumption levels, however, the first one would have a longer execution time than the last one.

9.4.2 Theoretical energy model

This section explains how to create theoretical energy models for different algorithms from a general perspective. We then apply this knowledge in Section 9.4.3 to create a specific model for the energy consumed by the VFDT.

The energy consumed by an algorithm can be estimated by identifying the type of events in the algorithm, e.g. floating point calculation, memory accesses, etc. We propose the following steps:

1. Identify the main functions of the algorithm
2. Define the type of events that are of interest, i.e. memory accesses, cache misses, floating point operations, and integer point operations. (Such as Eq. (9.8))
3. Map the different algorithm functions to the type of events, to end up with an equation based on the number of memory accesses and number of computations. (Such as Eqs. (9.12) and (9.15))
4. If needed, characterize the amount of energy consumed per type of event for that specific processor, based on the work by [31].

This approach can be adapted to any algorithm, and can thus provide insights into the energy behavior of an algorithm. The model is independent of programming language, etc., and focuses on basic operations in a particular algorithm, including access to resources such as data and memory. One of the main objectives with the model is to gain an understanding of which parts of the algorithms are most energy consuming.

In practice, when we measure the energy consumption on a real system this can be done in two ways: externally, i.e., we measure the current etc. consumed by the hardware, or internally, i.e., we measure how the software behaves. Most internal energy measurement / estimation tools work similarly, i.e., they count a number of hardware events using performance counters in the processors. These numbers are then fed into an energy model (similar to our simplified model), and then an estimation of the energy consumption is done. The RAPL framework by Intel that we use in our paper works in this way.

9.4.3 VFDT energy model

The energy model of the VFDT is based on the different steps to create an energy model from Section 9.4.2. The functions are taken from the pseudocode of the VFDT [1]. Alg. 6 shows the pseudocode for the VFDT algorithm with the *nmin adaptation* functionality added, but the main functions can also be observed there. The main functions are the following:

- **Sort instance to leaf**: When an instance is read, the first step is to traverse the tree based on the attribute values of that instance, to reach the correspondent leaf.

- **Update attributes**: Once the leaf is reached, the information at that leaf is updated with the attribute/class information of the instance. The update process is different if the attribute is numerical or nominal. For nominal attributes a simple table with the counts is needed. For updating the numerical attribute the mean and the standard deviation are updated.

- **Update instance count**: After each instance is read the counter at that leaf is updated.

- **Calculate entropy**: Once *nmin* instances are observed at a leaf, the entropy (information gain in this case) is calculated for each attribute.

- **Get best attribute**: The attributes with the highest information gain are chosen.

- **Calculate Hoeffding Bound**: We then compare the difference between the best and the second best attribute with the Hoeffding bound, calculated with Eq.(9.1).

- **Create new node**: If there is a clear attribute to split on, we split on the best attribute creating a new node.

Based on the information provided above, we present the energy consumption of the VFDT in the following model:

$$E_{VFDT} = E_{comp} + E_{cache\_tot} + E_{cache\_miss\_tot}, \tag{9.8}$$

where E_{comp} is the energy consumed on computations, E_{cache_tot} is the energy consumed on cache accesses, and $E_{cache\_miss\_tot}$ is the energy consumed on cache misses. They are defined as follows:

$$E_{comp} = n_{FPU} \cdot E_{FPU} + n_{INT} \cdot E_{INT}, \tag{9.9}$$

where n_{FPU} is the number of floating point operations, E_{FPU} is the average energy per floating point operation, n_{INT} is the number of integer operations, and E_{INT} is the average energy per integer operation.

$$E_{cache\_tot} = n_{cache} \cdot E_{cache}, \tag{9.10}$$

where n_{cache} is the number of accesses to cache, and E_{cache} is the average energy per access to cache. Finally,

$$E_{cache\_miss\_tot} = n_{cache\_miss} \cdot (E_{cache\_miss} + E_{DRAM}), \tag{9.11}$$

where n_{cache_miss} is the number of cache misses, E_{DRAM} is the average energy per DRAM access, and E_{cache_miss} is the average energy per cache miss.

The next step is to map these n_{FPU}, n_{INT}, n_{cache}, and n_{cache_miss} to the VFDT algorithm's functions, explained at the beginning of this section.

$$\begin{aligned} n_{FPU} = {} & n_{comp}(updating\_numerical\_atts) \\ & + n_{comp}(calc\_entropy) \\ & + n_{comp}(calc\_hoeff\_bound) \\ & + n_{comp}(get\_best\_att) \end{aligned} \tag{9.12}$$

$$\begin{aligned} n_{INT} = {} & n_{comp}(updating\_nominal\_atts) \\ & + n_{comp}(updating\_instance\_count), \end{aligned} \tag{9.13}$$

where $n_{comp}(f_i)$ refers to the number of computations required by function f_i.

$$n_{cache} = n_{acc}(updating\_atts) \tag{9.14}$$

$$\begin{aligned} n_{cache\_miss} = {} & n_{acc}(sorting\_instance\_to\_leaf) \\ & + n_{acc}(updating\_atts) \\ & + n_{acc}(calc\_entropy) \\ & + n_{acc}(calc\_hoeff\_bound) \\ & + n_{acc}(new\_node), \end{aligned} \tag{9.15}$$

where $n_{acc}(f_i)$ represents the number of accesses to memory or cache in order to execute function f_i. The number of cache and memory accesses of updating the attributes ($updating\_atts$) will depend on the block size of the cache. If the block size is big enough, then we would have one cache miss to update the information of the first attribute, and then cache hits

for the rest of the attributes. However, if there are many attributes, thus not fitting on the block size B, then there will be a cache miss for every attribute that exceeds the block size. We also consider the presence of a cache miss every time a node of the tree is traversed, and every time we calculate the entropy and Hoeffding bound values.

The last step is to express these number of accesses and computations based on the number of instances (N), the $nmin$ value, the number of numerical attributes (A_f), the number of nominal attributes (A_i), and the block cache size B. We then obtain the following:

$$\begin{aligned} n_{FPU} &= N \cdot A_f + \frac{N}{nmin} \cdot (A_f + A_i) \\ &+ \frac{N}{nmin} + \frac{N}{nmin} \cdot (A_f + A_i) \\ &= N \cdot A_f + 2 \cdot \frac{N}{nmin} \cdot (A_f + A_i) + \frac{N}{nmin} \end{aligned} \tag{9.16}$$

Updating numerical attributes is one access per instance per numerical attribute; calculating the entropy is one access per attribute (thus the sum of nominal and numerical attributes) every $nmin$ instances; calculating the Hoeffding bound is one access every $nmin$ instances; and calculating the best attribute is the same as calculating the entropy.

$$n_{INT} = N \cdot A_i + N \tag{9.17}$$

Updating nominal attributes is, as before, one access per instance per nominal attribute; and one access per instance for updating the counter.

$$n_{cache} = N \cdot (A_f + A_i - \frac{A_f + A_i}{B}) \tag{9.18}$$

To update the attributes, we consider one cache hit per all attributes per instance, minus all the attributes that don't fit on the block size B and create cache misses.

$$
\begin{aligned}
n_{cache\_miss} &= N \cdot (A_f + A_i + \frac{A_f + A_i}{B}) \\
&+ \frac{N}{nmin} + \frac{N}{nmin} + \frac{N}{nmin} \\
&= N \cdot (A_f + A_i + \frac{A_f + A_i}{B}) + 3 \cdot \frac{N}{nmin}
\end{aligned}
\tag{9.19}
$$

To calculate the number of accesses of sorting an instance to a leaf we assume that we need to access one level per attribute, which is the worst case scenario. So the total number of accesses in this case is one per instance per attribute. To update the attributes, as was explained before, it's one miss per all attributes that exceed the block size B, per instance. To access the needed values to calculate the entropy, the Hoeffding bound, and to split, we consider one access every $nmin$ instances.

Based on Eqs. (9.8), (9.9), (9.10), (9.11), (9.16), (9.17), (9.18), and (9.19), our final energy model equation is the following:

$$
\begin{aligned}
E_{VFDT} = E_{FPU} &\cdot (N \cdot A_f + 2 \cdot \frac{N}{nmin} \cdot (A_f + A_i) \\
&+ \frac{N}{nmin}) + E_{INT} \cdot (N \cdot A_i + N) \\
&+ E_{cache} \cdot (N \cdot (A_f + A_i - \frac{A_f + A_i}{B})) \\
&+ (E_{cache\_miss} + E_{DRAM}) \cdot (N \cdot (A_f + A_i \\
&+ \frac{A_f + A_i}{B}) + 3 \cdot \frac{N}{nmin})
\end{aligned}
\tag{9.20}
$$

This is a general and simplified model of how the VFDT algorithm consumes energy. The energy values (i.e. E_{cache}, E_{FPU}, E_{INT}, E_{DRAM}, and E_{cache_miss}) will vary depending on the processor and architecture, although there is a lot of research that ranks these operations based on their energy consumption [31]. For instance, a DRAM instruction consumes three orders of magnitude more energy than an ALU operation. We can see the importance of the number of attributes in the overall energy consumption of the algorithm. Since E_{FPU} is significantly higher than E_{INT}, numerical attributes have a higher impact on energy consumption than nominal attributes.

9.5 Experimental Design

In comparison to our previous work [4], we have designed extensive experiments to better understand the behavior of VFDT, VFDT-*nmin*, and CVFDT (Concept-Adapting Very Fast Decision Tree [5]) in three setups:

- Baseline
- Concept Drift
- Real World

The baseline setup presents a sensitivity analysis where we evaluate the accuracy and energy consumption of the mentioned algorithms while varying the input parameters of the dataset. In particular, we vary the number of instances, nominal, and numerical attributes, to understand how that affects energy consumption and accuracy. We have already observed through our energy model that the number of numerical attributes affected significantly the energy consumption. This setup aims at validating empirically that observation, to be used as a baseline to the other experiments.

The concept drift setup investigates the effect of concept drift in accuracy and energy consumption. We have taken three synthetically generated datasets, LED, RBF, and waveform, and added two levels of change.

The real world setup investigates the energy consumption and accuracy of the VFDT, VFDT-*nmin*, and CVFDT, in six real datasets.

The datasets used in our experiments are explained, per setup, in Table 9.1, and described in Section 9.5.1. We run the experiments on a machine with an 3.5 GHz Intel Core i7, with 16GB of RAM, running OSX. To estimate the energy consumption we use Intel Power Gadget[1], that accesses the performance counters of the processor, together with Intel's RAPL interface to obtain energy consumption estimations [32]. The implementation of VFDT-*nmin* together with the scripts to conduct the experiments are publicly available[2].

[1] `https://software.intel.com/en-us/articles/intel-power-gadget-20`
[2] `https://github.com/egarciamartin/hoeffding-nmin-adaptation`

9.5.1 Datasets

We have used synthetic datasets for the baseline and concept drift setup, and six different real datasets for the last setup. The choice of datasets is inspired by the work of [33]. The datasets are described in Table 9.1. There are a total of 29 datasets, 23 artificial datasets generated with Massive Online Analysis (MOA) [34], and 6 real world datasets. The artificial datasets are listed in Table 9.1.

Table 9.1: *Datasets used in the experiment to compare VFDT, VFDT-nmin, and CVFDT. A_i and A_f represent the number of nominal and numerical attributes, respectively. The details of each dataset is presented in Section 9.5.1*

Dataset	Train	Test	A_i	A_f	Class
Baseline					
RT_10k_10_10	6,700	3,300	0	10	2
RT_100k_10_10	67,000	33,000	0	10	2
RT_1M_10_10	670,000	330,000	0	10	2
RT_10M_10_10	6,700,000	3,300,000	0	10	2
RT_1M_10_0	670,000	330,000	10	0	2
RT_1M_20_0	670,000	330,000	20	0	2
RT_1M_30_0	670,000	330,000	30	0	2
RT_1M_40_0	670,000	330,000	40	0	2
RT_1M_50_0	670,000	330,000	50	0	2
RT_1M_0_10	670,000	330,000	0	10	2
RT_1M_0_20	670,000	330,000	0	20	2
RT_1M_0_30	670,000	330,000	0	30	2
RT_1M_0_40	670,000	330,000	0	40	2
RT_1M_0_50	670,000	330,000	0	50	2
Concept Drift					
LED	670,000	330,000	24	0	10
LED_3	670,000	330,000	24	0	10
LED_7	670,000	330,000	24	0	10
RBF	670,000	330,000	0	10	2
RBF_m	670,000	330,000	0	10	2
RBF_f	670,000	330,000	0	10	2
waveform	670,000	330,000	0	21	3
waveform_5	670,000	330,000	0	21	3
waveform_10	670,000	330,000	0	21	3
Real World					
airline	539,383	99,999	4	3	2
electricity	30,359	14,953	1	6	2
poker	555,564	273,637	5	5	10
CICIDS	461,802	230,901	78	5	6
forest	387,342	193.670	40	10	7
kddcup	3,265,621	1,632,810	7	34	23

RT_*inst*_A_i_A_f: Random tree dataset with *inst* number of instances, A_i number of nominal attributes, and A_f number of numerical attributes. This dataset is inspired from the dataset proposed by the authors of the original VFDT [1]. It first builds the tree, by randomly selecting attributes to split, assigning random values to the leaves. The leaves will be the classes of the instances. Then new examples are generated, with random attribute values, and they are labeled based on the already created tree.

LED_x: LED dataset with x attributes with drift. The goal is to predict the digit on a LED display with seven segments, where each attribute has a 10% chance of being inverted [35].

RBF_v: The radial based function (RBF) dataset has 10 numerical attributes. The generator creates n number of centroids, each with a random center, class label and weight. Each new example randomly selects a center, considering that centers with higher weights are more likely to be chosen. The chosen centroid represents the class of the example. Drift is introduced by moving the centroids with speed v, either moderate (0.001), or fast (0.01). More details are given by [33].

waveform_x: Waveform dataset with x attributes with drift. The waveform dataset comes from the UCI repository. The function generates a wave as a combination of two or three base waves. The task is to differentiate between the three waves.

We have also tested six real datasets, some of them available from the MOA official website[3]. The poker dataset is a normalized dataset available from the UCI repository. Each instance represents a hand consisting of five playing cards, where each card has two attributes; suit and rank. The electricity dataset is originally described in [36], and is frequently used in the study of performance comparisons. Each instance represents the change of the electricity price based on different attributes such as day of the week, represented by the Australian New South Wales Electricity Market. The airline dataset [37] predicts if a given flight will be delayed based on attributes such as airport of origin and airline. The forest dataset *contains the forest cover type for 30 x 30 meter cells obtained from US Forest Service*

[3] `https://moa.cms.waikato.ac.nz/datasets/`

(USFS) Region 2 Resource Information System (RIS) data[4]. The KDDCUP dataset[5][38] was created for a competition in 1999, where the goal was to detect network intrusions. A similar but newer data is the CICIDS [39], a dataset in cybersecurity where the task is again to detect intrusions.

9.5.2 Algorithms

We compare VFDT, VFDT-*nmin*, and CVFDT under the mentioned datasets. The initial value of *nmin* is set to 200, which is the default value used by the authors of the VFDT. We evaluate all algorithms based on the following measures: accuracy (percent of correctly classified instances), energy consumed by the processor, and energy consumed by the DRAM. We evaluate the accuracy by having a training set and a test set that is different from the training set, as can be observed in Table 9.1. We have not performed yet prequential evaluation as with this method, however that is planned for future works.

9.5.3 Statistical Significance

To test if the differences between accuracy and energy consumption between the VFDT and the VFDT-*nmin* are statistically significant, we perform a parametric test, namely the dependent sample t-test [40].

We choose a parametric test after having tested for normality between the differences in accuracy and energy consumption between the VFDT and VFDT-*nmin*, obtaining p-values smaller than 0.01. We choose this test in particular because the observations are paired based on the dataset, thus can be considered as dependent.

We first test if the VFDT-*nmin* obtained significantly higher accuracy than the VFDT, since the data indicates that the average of the accuracy of the VFDT-*nmin* is 1.41% higher than for he VFDT. Thus, we propose the following null and one-tailed alternative hypothesis [41]:

$\boldsymbol{H_0} : \mu_{A1} = \mu_{A2}$, where μ_{A1} represents the mean of accuracy values for the VFDT, and μ_{A2} represents the mean of accuracy values for VFDT-*nmin*. Thus, the null hypothesis states that the means of the accuracy values between the VFDT and the VFDT-*nmin* are equal.

[4] `https://moa.cms.waikato.ac.nz/datasets/`

[5] `http://kdd.ics.uci.edu/databases/kddcup99/task.html`

$\boldsymbol{H_1} : \mu_{A1} < \mu_{A2}$, stating that the mean of the accuracy of VFDT-*nmin* is higher than the mean of the accuracy of the VFDT.

If the *p-value*, obtained as a result of the test, is lower than the chosen alpha level (0.01 for this paper), we can conclude that the VFDT-*nmin* obtains significantly higher accuracy than the VFDT.

We perform the same statistical tests to check if the difference in energy consumption between the VFDT and the VFDT-*nmin* are statistically significant. After testing for normality, and obtaining a *p-value* of less than 0.01, we choose to perform the same parametric test as before, paired student t-test. The null and alternative hypothesis are the following:

$\boldsymbol{H_0} : \mu_{E1} = \mu_{E2}$, where μ_{E1} represents the mean of energy consumption values for the VFDT, and μ_{E2} represents the mean of energy consumption values for VFDT-*nmin*. Thus, the null hypothesis states that the VFDT and the VFDT-*nmin* consume equal amounts of energy.

$\boldsymbol{H_1} : \mu_{E1} > \mu_{E2}$, stating that the mean of the energy consumed by the VFDT is higher than the mean of the energy consumed by the VFDT-*nmin*. Since we employ directional alternative hypothesis, the null hypothesis can only be rejected if the data indicates that the mean of the energy consumed by the VFDT is significantly higher than the mean of the energy consumed by the VFDT-*nmin*.

9.6 Results and Discussion

The results of the experiments are divided in the three setups defined above. Tables 9.4, 9.5, and 9.6 present the accuracy and energy consumption results of the *baseline, concept drift,* and *real datasets* setups respectively. We have evaluated the accuracy as the percentage of correctly classified instances, and the energy consumption as the energy estimated by the Intel Power Gadget tool, summing the energy consumed by the processor and the DRAM to obtain the total energy consumption. We have run the experiments 5 times and averaged the results. Table 9.2 summarizes the energy and accuracy results from all datasets by showing the difference between VFDT-*nmin* and VFDT; and between VFDT-*nmin* and CVFDT.

We initially discuss the statistical significance results performed on the accuracy and energy consumption data from the VFDT and the VFDT-

Table 9.2: *Difference in accuracy (ΔAcc) and energy consumption (ΔEnergy) between VFDT and VFDT-nmin; and VFDT-nmin and CVFDT. A positive number in accuracy means that VFDT-nmin obtained a higher accuracy. A negative number in energy means that the VFDT-nmin reduced the energy consumption by that percentage. Higher accuracy and lower energy consumption of the VFDT-nmin are presented in bold*

	VFDT-nmin vs VFDT		VFDT-nmin vs CVFDT	
Dataset	ΔAcc (%)	ΔEnergy(%)	ΔAcc (%)	ΔEnergy(%)
Baseline				
RT_10K_10_10	**0.00**	**-12.35**	**1.91**	**-66.44**
RT_100K_10_10	**0.00**	**-2.44**	-9.67	**-82.23**
RT_1M_10_10	-0.33	**-4.77**	**3.91**	**-89.86**
RT_10M_10_10	-0.31	**-1.70**	**6.60**	**-84.02**
RT_1M_0_10	-0.25	**-0.26**	**1.41**	**-94.05**
RT_1M_0_20	-0.20	**-2.75**	**1.39**	**-93.33**
RT_1M_0_30	-0.03	**-2.04**	**1.04**	**-95.41**
RT_1M_0_40	**0.05**	**-1.47**	**1.19**	**-96.29**
RT_1M_0_50	-0.06	**-1.37**	**0.98**	**-97.04**
RT_1M_10_0	**0.73**	**-1.15**	**11.63**	**-84.15**
RT_1M_20_0	-1.65	**-1.19**	**9.96**	**-84.50**
RT_1M_30_0	**3.88**	2.04	**17.84**	**-83.98**
RT_1M_40_0	**5.13**	0.78	**16.34**	**-85.40**
RT_1M_50_0	**25.28**	0.33	**42.80**	**-84.29**
Concept Drift				
LED	**1.78**	2.21	**2.25**	**-82.19**
LED_3	**1.78**	1.14	**2.25**	**-81.42**
LED_7	**1.78**	0.33	**2.25**	**-81.47**
RBF	-0.89	**-11.72**	**1.16**	**-93.85**
RBF_m	-0.29	**-19.96**	**0.63**	**-91.18**
RBF_f	**0.28**	**-19.95**	**1.45**	**-95.80**
waveform	-1.09	**-24.92**	**1.26**	**-87.18**
waveform_10	-0.85	**-21.87**	**1.49**	**-86.74**
waveform_5	-0.85	**-21.98**	**1.49**	**-86.81**
Real				
CICIDS17	**0.18**	**-3.70**	**2.09**	**-89.84**
airline	**0.07**	**-11.19**	**11.97**	**-87.39**
electricity	**3.32**	**-31.61**	**5.10**	**-77.07**
forest	**0.20**	**-2.47**	**3.19**	**-83.36**
kddcup	**0.00**	**-5.82**	**39.62**	**-82.80**
poker	**3.17**	8.81	**16.98**	**-82.37**
Average	1.41	-6.59	6.91	-86.57

nmin. We then discuss the difference in energy consumption and accuracy from the baseline datasets between VFDT, VFDT-*nmin*, and CVFDT. The aim is to have a general understanding of the algorithm in terms of energy

Table 9.3: *Results from performing a student t-test on the differences in accuracy and energy consumption between the VFDT and VFDT-nmin on all datasets*

Measure	p-value	Null Hypothesis
Accuracy	0.0154	Not rejected. Higher than 0.01
Energy	0.0017	Rejected. Lower than 0.01

Table 9.4: *Baseline setup. Energy consumption and accuracy results of varying the number of nominal attributes, numerical attributes, and instances. RT_inst_A_i_A_f, where inst is the number of instances, A_i is the number of nominal attributes and A_f the number of numerical attributes. Algorithms: VFDT, VFDT-nmin, and CVFDT. Measurements: Accuracy, Total energy. Total energy = CPU energy + DRAM energy. Higher accuracy and lower total energy consumption values per setup are presented in bold.*

Dataset	Accuracy (%)			Total Energy(J)		
	vfdt-nmin	cvfdt	vfdt	vfdt-nmin	cvfdt	vfdt
# Instances						
RT_10K_10_10	**66.42**	64.52	**66.42**	**3.52**	10.49	4.02
RT_100K_10_10	**74.45**	**84.12**	74.45	**53.58**	301.52	54.92
RT_1M_10_10	94.31	90.40	**94.64**	**537.88**	5303.19	564.81
RT_10M_10_10	98.09	91.49	**98.41**	**10063.04**	62963.14	10237.39
# Nominal Att						
RT_1M_10_0	**97.07**	85.44	96.35	**159.90**	1008.99	161.75
RT_1M_20_0	79.60	69.64	**81.25**	**268.46**	1731.49	271.68
RT_1M_30_0	**74.78**	56.93	70.90	382.26	2386.39	**374.63**
RT_1M_40_0	**95.38**	79.05	90.25	494.98	3389.69	**491.15**
RT_1M_50_0	**98.24**	55.44	72.96	604.05	3843.90	**602.05**
# Numerical Att						
RT_1M_0_10	99.39	97.98	**99.64**	**408.95**	6867.78	410.03
RT_1M_0_20	99.36	97.97	**99.56**	**723.50**	10847.07	743.93
RT_1M_0_30	99.26	98.21	**99.28**	**1060.73**	23129.46	1082.80
RT_1M_0_40	**99.44**	98.26	99.40	**1443.50**	38868.37	1465.11
RT_1M_0_50	99.22	98.24	**99.28**	**1777.81**	60079.04	1802.47

consumption, and how the number of nominal and numerical attributes affect it. We further compare how concept drift affects both accuracy and energy consumption in different datasets with different levels of concept drift. Finally we compare how well the *nmin adaptation* method works in real world datasets, in terms of accuracy and energy consumption.

Table 9.5: *Energy consumption and accuracy results of concept drift datasets. For the LED and waveform datasets, the number after the _ represents the number of attributes with drift. For the RBF dataset, the f represents fast drift (speed of 0.01) and m moderate drift (speed of 0.001). Algorithms: VFDT, VFDT-nmin, and CVFDT. Measurements: Accuracy, Total energy. Total energy = CPU energy + DRAM energy. Higher accuracy and lower total energy consumption values for each dataset are presented in bold.*

Dataset	Accuracy (%)			Total Energy(J)		
	vfdt-nmin	cvfdt	vfdt	vfdt-nmin	cvfdt	vfdt
LED	**72.77**	70.52	70.99	240.20	1348.48	**235.02**
LED_3	**72.77**	70.52	70.99	242.19	1303.67	**239.46**
LED_7	**72.77**	70.52	70.99	241.75	1304.80	**240.97**
RBF	89.01	87.85	**89.90**	**529.95**	8622.95	600.31
RBF_f	**53.05**	51.60	52.76	**512.69**	12201.72	640.45
RBF_m	50.67	50.04	**50.96**	**521.68**	5915.54	651.81
waveform	78.04	76.79	**79.14**	**929.01**	7248.51	1237.29
waveform_10	78.28	76.79	**79.12**	**946.79**	7141.90	1211.88
waveform_5	78.28	76.79	**79.12**	**949.20**	7198.39	1216.65

Table 9.6: *Energy consumption and accuracy results of real world datasets. Algorithms: VFDT, VFDT-nmin, and CVFDT. Measurements: Accuracy, Total energy. Total energy = CPU energy + DRAM energy. Higher accuracy and lower total energy consumption values for each dataset are presented in bold.*

Dataset	Accuracy (%)			Total Energy(J)		
	vfdt-nmin	cvfdt	vfdt	vfdt-nmin	cvfdt	vfdt
CICIDS17	**98.11**	96.02	97.94	**1041.62**	10252.09	1081.70
airline	**67.01**	55.04	66.94	**89.47**	709.53	100.74
electricity	**76.23**	71.13	72.91	**7.24**	31.56	10.58
forest	**59.99**	56.80	59.79	**341.77**	2054.04	350.42
kddcup	**87.03**	47.41	**87.03**	**3698.26**	21497.73	3926.66
poker	**75.44**	58.46	72.27	142.45	807.87	**130.91**

9.6.1 Statistical Characterization

This section presents the results of the statistical significance analysis between the VFDT and the VFDT-*nmin*. We omit the comparison against the CVFDT since the differences are clear, and CVFDT obtains significantly higher energy consumption on all datasets, and worse accuracy on all except one.

We have performed a paired student t-test on the data, with a confidence

level of 0.01. The p-values of the tests are presented in Table 9.3.

The results show that there is no statistically significant difference between the accuracy of the VFDT and the VFDT-*nmin*, since the p-value is higher than 0.01. Thus, the null hypothesis, which stated that the means of the accuracy of both algorithms are equal (Section 9.5.3), can not be rejected. The results also show that there is a statistical difference between the energy consumption values between the VFDT and the VFDT-*nmin*. The null hypothesis can be rejected, since the p-value (0.0017) is lower than 0.01. Thus, the alternative hypothesis is supported, which stated that the energy consumed by the VFDT-*nmin* is significantly lower than the energy consumed by the VFDT.

9.6.2 Baseline Setup

The baseline setup (Table 9.4) is done by varying the number of instances, nominal and numerical attributes from the random tree dataset. Fig. 9.4 compares the amount of accuracy and energy consumed when the number of nominal attributes are varied between 10, 20, 30, 40, and 50. We observe that the VFDT and VFDT-*nmin* behave very similarly, i.e., the higher the number of attributes the higher the energy consumption. However, the CVFDT has a significantly higher increase in energy consumption when increasing the number of attributes. Based on this analysis, we can conclude that the CVFDT does not scale with the number of attributes. Fig. 9.4 also shows the accuracy per number of nominal attributes, where we noticed a strange phenomenon. VFDT-*nmin* obtains significantly higher levels of accuracy than the VFDT. This is visible especially for the dataset with 50 nominal attributes. We believe that this may be due to some error with the code that was provided in the original implementation. We have tested the same dataset on the new version from MOA [34] and it has not shown such behavior.

Fig. 9.5 shows the same comparison as Fig. 9.4 but for numerical attributes. We can see a similar pattern in terms of energy consumption. Namely, the higher the number of numerical attributes the higher the energy consumption. While VFDT-*nmin* and VFDT show a linear increase, CVFDT shows an exponential increase. The latter confirms our previous claim that it is non scalable. However, in terms of accuracy, all algorithms behave as expected. The accuracy is almost identical, being independent of the number

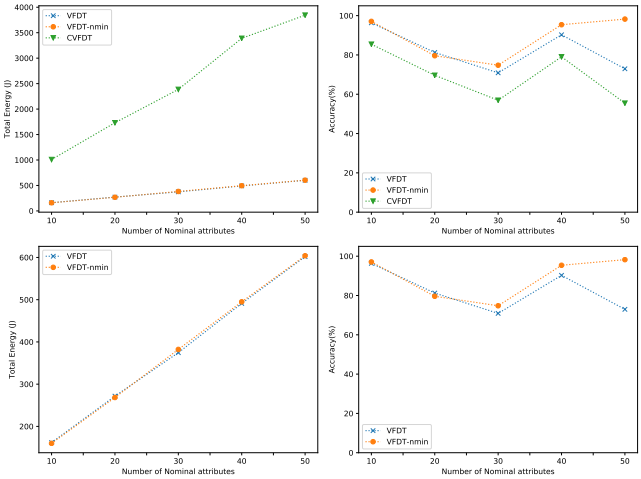

Figure 9.4: *Baseline setup. Comparison on energy consumption and accuracy of the number of nominal attributes for the VFDT-nmin, VFDT, and CVFDT. The lower figures are expanded version of the upper figures (without CVFDT)*

of attributes. Fig. 9.6 compares the energy consumption of the nominal and numerical attributes by averaging all algorithms (VFDT, VFDT-*nmin*, and CVFDT). The figures empirically validates what we theoretically claimed with the energy model in Section 9.4.3. Namely, that handling numerical attributes results in a higher energy cost compared to handling nominal attributes. We can observe that handling 50 numerical attributes costs 12.6X more than handling 50 nominal attributes (averaged for VFDT-*nmin*, VFDT, and CVFDT).

We have studied the energy consumption and accuracy of varying the number of instances between 10k, 100k, 1M, and 10M. Fig. 9.7 shows that there is not a linear increase between the number of instances and the energy consumption. When the number of instances reaches 10M, the energy consumption increases by 18X (compared to 1M), for the VFDT-*nmin* and VFDT; and by 12X for the CVFDT. The accuracy increases with the number of instances, which is the expected behavior, since with more instances the tree is able to reflect better the dataset.

From Table 9.4 we can observe that the VFDT-*nmin* obtained the lowest energy consumption in 11 out of 14 datasets. This is achieved while obtaining either the highest accuracy, or up to a 1.65% lower compared to

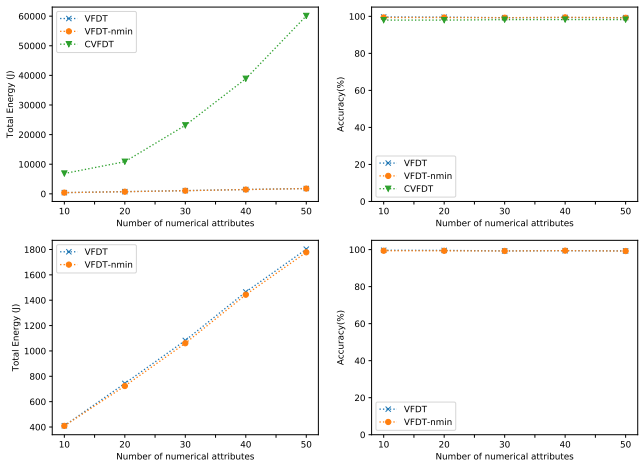

Figure 9.5: *Baseline setup. Comparison on energy consumption and accuracy of the number of numerical attributes for the VFDT-nmin, VFDT, and CVFDT. The lower figures are expanded version of the upper figures (without CVFDT)*

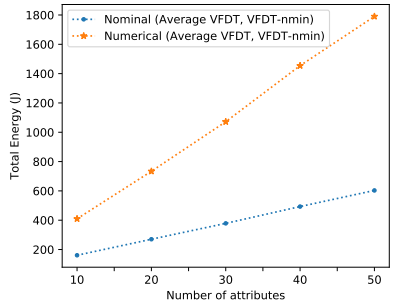

Figure 9.6: *Baseline setup. Comparison of the nominal and numerical attributes on energy consumption and accuracy for the VFDT-nmin, VFDT, and CVFDT. The lower figures are expanded version of the upper figures (without CVFDT)*

the algorithm with the highest accuracy. The CVFDT obtains the highest energy consumption values for all datasets, and it only obtains the highest accuracy in one of the datasets. We conclude that, for the baseline datasets, the VFDT-*nmin* obtains significantly lower energy at a cost of less than 1% of accuracy on average.

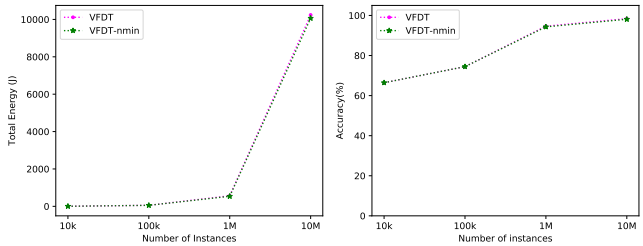

Figure 9.7: *Baseline setup. Comparison on energy consumption and accuracy of the number of instances for the VFDT-nmin, VFDT, and CVFDT. The lower figures are expanded version of the upper figures (without CVFDT)*

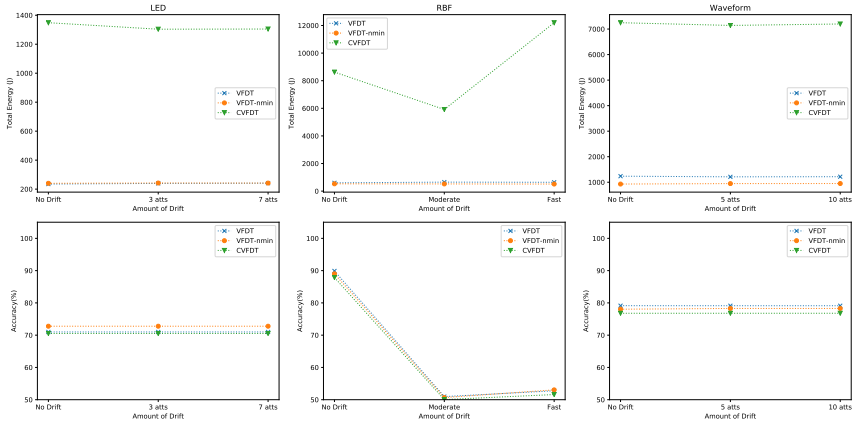

Figure 9.8: *Concept drift setup. Comparison on energy consumption and accuracy for the LED, RBF, and waveform datasets for the VFDT-nmin, VFDT, and CVFDT.*

9.6.3 Concept Drift Setup

Figs. 9.8 and 9.9 show the energy consumption and accuracy results of running the VFDT, VFDT-*nmin*, and CVFDT algorithms on datasets with different levels of concept drift (see also Table 9.5). Fig. 9.9 focuses on VFDT and VFDT-*nmin* for a clearer view, since CVFDT has high levels of energy consumption and does not allow to view the difference in energy between VFDT and VFDT-*nmin*. As we already observed with the baseline setup, CVFDT consumes significantly more energy than VFDT and VFDT-*nmin*. However, in terms of accuracy, it obtains a lower accuracy in all of the datasets with concept drift. This behavior is not expected, since the

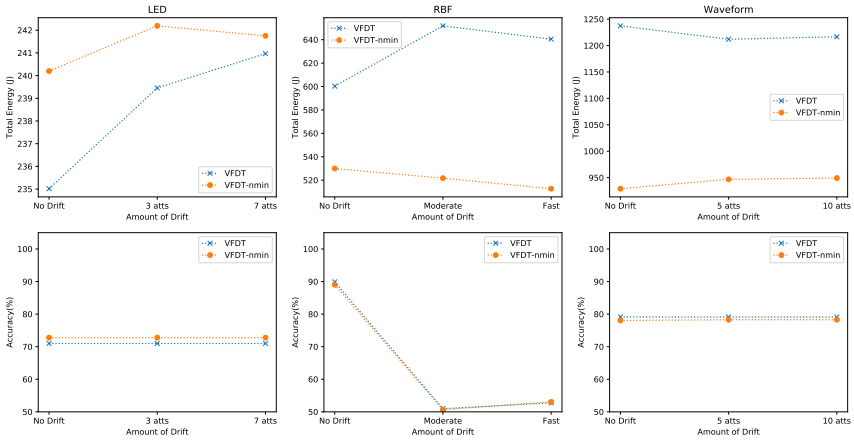

Figure 9.9: *Concept drift setup. Comparison on energy consumption and accuracy for the LED, RBF, and waveform datasets for the VFDT-nmin, and VFDT.*

CVFDT is supposed to handle concept drift, but even in datasets with a high level of drift, such as RBF fast, it obtains lower accuracy than both VFDT and VFDT-*nmin*. Taking a look at Fig. 9.9, we observe how VFDT-*nmin* obtains significantly lower energy consumption than VFDT for the RBF and waveform datasets. On the other hand, VFDT-*nmin* obtains slightly higher energy consumption for the LED dataset, but it seems to scale better when increasing the number of attributes with drift. VFDT-*nmin* decreases energy consumption when instead of having 3 attributes with drift we have 7 (higher drift), while VFDT increases in energy consumption. This phenomenon is also visible for the RBF dataset, where VFDT-*nmin* scales in terms of energy consumption when increasing drift, and VFDT does not.

Moreover, all three algorithms decrease accuracy for the RBF dataset when increasing the amount of drift, although there is a slight increase when moving from moderate to fast drift. The LED and waveform dataset give almost the same accuracy when varying the amount of drift. The reason for this is that the algorithm is able to learn the data even with drift, and there is no possibility for a higher accuracy in these datasets (with this type of algorithms). VFDT-*nmin* obtains higher accuracy for the LED dataset independently on the amount of drift, but at a cost of higher energy consumption. All in all, VFDT-*nmin* obtains significantly lower energy consumption on 6/9 datasets. Moreover, VFDT-*nmin* scales better than

VFDT in terms of energy consumption when increasing the amount of drift or the number of attributes with drift. VFDT and VFDT-*nmin* obtain very similar levels of accuracy (less than 1% difference in average), while VFDT-*nmin* even obtains a higher accuracy in 3/9 datasets. As for other the other datasets, CVFDT obtains significantly higher energy consumption on all drift datasets while not giving improvements in accuracy.

9.6.4 Real World Setup

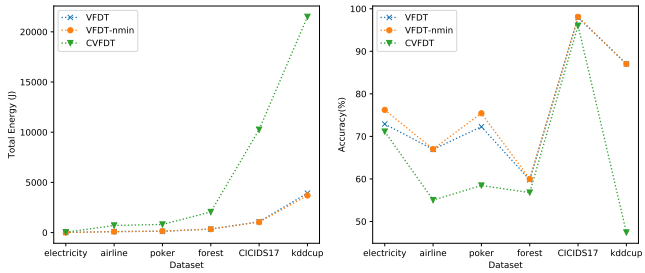

Figure 9.10: *Real world setup. Results of running the VFDT, VFDT-nmin, and CVFDT on real world datasets. The datasets are sorted by the energy consumption (ascending) of the VFDT-nmin*

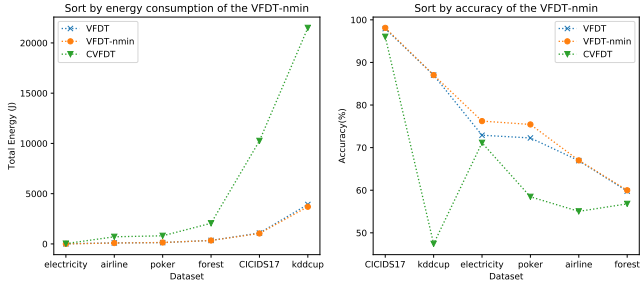

Figure 9.11: *Real world setup. Results of running the VFDT, VFDT-nmin, and CVFDT on real world datasets. The datasets are sorted by the energy consumption (ascending) of the VFDT-nmin on the left plot, and by the accuracy (descending) of the VFDT-nmin on the right plot*

This last section describes the energy consumption and accuracy results of running VFDT, VFDT-*nmin*, and CVFDT in six real world datasets (explained in Section 9.5.1). As we can see from Table 9.6, VFDT-*nmin* obtains

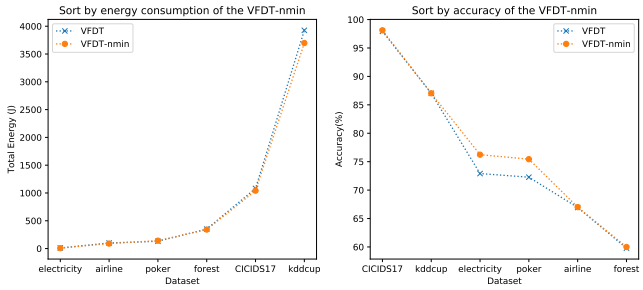

Figure 9.12: *Real world setup. Results of running the VFDT, VFDT-nmin, on real world datasets. The datasets are sorted by the energy consumption of the VFDT-nmin on the left plot, and by the accuracy of the VFDT-nmin on the right plot*

the highest accuracy on all datasets, and the lowest energy consumption on 5/6 datasets. The differences in accuracy between VFDT and VFDT-*nmin* are in average of 1.16%. For the electricity dataset, VFDT-*nmin* obtains the maximum accuracy gain, of 3.3% compared to VFDT. Similar to the previous setups, CVFDT obtains significantly lower levels of accuracy on all datasets at a significantly higher energy cost.

This is clear from Figs. 9.10, 9.11, and 9.12. Fig. 9.12 restricts the results to the VFDT and VFDT-*nmin* for a clear view. Fig. 9.10 shows how there is not a clear direct connection between a higher energy consumption and a higher accuracy. Taking a look at the forest dataset, for instance, we see how this dataset consumes more energy than the electricity, airline, and poker datasets, but it obtains a lower accuracy than all of them. KDDcup is the dataset with the highest energy consumption because it is also the dataset with the highest number of instances. In relation to how the number of attributes impact the energy consumption, we observe how CICIDS obtains significantly higher energy consumption than the poker dataset, while poker has more number of instances than CICIDS. The reason is because CICIDS has 73 more attributes than poker.

9.6.5 Summary

The results of the paper are summarized below:

- VFDT-*nmin* obtains significantly lower levels of energy consumption

in comparison to VFDT for 22 out of the 29 studied datasets (76% of the datasets).

- VFDT and VFDT-*nmin* obtain on average less than 1% difference in accuracy. VFDT-*nmin* obtains higher accuracy than VFDT on 55% of the datasets.
- CVFDT obtains significantly higher energy consumption and lower accuracy in almost all datasets, even on concept drift datasets.
- Regarding the baseline setup, we can see how handling numerical attributes costs up to 12X more energy than handling nominal attributes. This matches with our theoretical claims from the energy model (Section 9.4.3). The reason for this difference in energy is because the average energy per floating point operation (E_{FPU}) is significantly higher than the average energy per integer instruction (E_{INT}), as explained by [31].
- We also observed that while VFDT and VFDT-*nmin* scale linearly in terms of energy consumption when increasing the number of attributes, CVFDT scales poorly with the increase of both number of attributes and instances.
- Regarding concept drift, we observed that VFDT-*nmin* scales better than VFDT in terms of energy consumption when increasing the amount of drift or the number of attributes with drift.
- VFDT-*nmin* obtained a higher accuracy compared to VFDT for all real world datasets, obtaining a significantly lower energy energy consumption in 83% of the datasets. We also observed how there was not a direct correlation between a higher energy consumption needed to obtain a higher accuracy. This was explained simply by the size and the dimension (number of attributes) of the dataset.

9.7 Limitations

There are a few limitations worth mentioning in this study. Regarding the *nmin adaptation* method, it adapts to the optimal nmin parameter making the assumption that the future observed data will follow the same distribution as the already observed data. Regarding the implementation

and code used, we are aware that there might be a bug in the original implementation of the VFDT, that is the one used in this study. This was mentioned in the baseline setup, since we obtained unexpected results for the datasets with high number of nominal attributes. We are going to use MOA, which has the newer implementation of the code, in future studies.

Regarding the energy consumption estimation, the energy consumption values are not given per program, but for the complete system (DRAM or CPU). Thus, we base our conclusions on the comparison between the algorithms and setups, since all of them are run on the same scenarios. Since estimating energy consumption in software is challenging [42], we have created the energy model presented in Sections 9.4.2 and 9.4.3, which is independent of the programming language and hardware platform. We understand that this is a simplified model of all the events present in the algorithm. However, it is a straightforward approach to understand, from a theoretical perspective, which parts of the algorithm are the most energy inefficient. More details on different approaches to estimate energy are giving in the already mentioned survey [42].

9.8 Conclusions

This paper introduced *nmin adaptation*, a method that extends standard Hoeffding trees to reduce their energy consumption. *nmin adaptation* allows for a faster growth on the branches with higher confidence to split, and delays the growth on the less confident branches. This reduces unnecessary computations, reducing energy consumption with only minor effects on accuracy.

This paper extends our previous work: "Hoeffding Trees with *nmin adaptation*" by adding extensive experiments that validate the proposed method with statistical tests. We have evaluated the accuracy and energy consumption of the VFDT, VFDT-*nmin* (VFDT with *nmin adaptation*), and CVFDT algorithms, under 29 public datasets. The results show that VDFT-*nmin* consumes up to 31% less energy, affecting accuracy at most by a 1.65%, in comparison with the standard VFDT. In comparison to CVFDT, VFDT-*nmin* consumes 85% less energy, obtaining 6% higher accuracy values, on average.

In particular, we have first conducted a sensitivity analysis to determine the energy consumption and accuracy of the mentioned algorithms when

varying the number of instances, nominal and numerical attributes. The results of this analysis show that handling numerical attributes can consume up to 12X more energy than handling nominal attributes. We then compared VFDT, VFDT-*nmin* and CVFDT on concept drift datasets. The results of this setup conclude that VFDT-*nmin* consumes significantly less energy consumption than the other two algorithms (13% on average compared to the VFDT, 87% compared to the CVFDT), while obtaining similar levels of accuracy. We finally compared the mentioned algorithms in six real world datasets. VFDT-*nmin* obtained higher accuracy than VDFT and CVFDT, while obtaining 7% less energy consumption than the VFDT and 84% less energy consumption than the CVFDT.

Low energy consumption is one of the key requirements for algorithms to be able to run in the edge (e.g. mobile and embedded devices). While data stream mining algorithms are designed to run in the edge, due to their high velocity and low memory usage, they still have not considered energy consumption. To address that gap, we present a method that allows for an energy efficient approach to design Hoeffding Trees, without affecting its predictive performance.

9.9 References

[1] P. Domingos and G. Hulten. "Mining high-speed data streams". In: *Proc. 6th SIGKDD Int'l Conf. on Knowledge discovery and data mining*. 2000, pp. 71–80.

[2] C. F. Rodrigues, G. Riley, and M. Luján. "Fine-grained energy profiling for deep convolutional neural networks on the Jetson TX1". In: *Workload Characterization (IISWC), 2017 IEEE International Symposium on*. IEEE. 2017, pp. 114–115.

[3] K. Gauen, R. Rangan, A. Mohan, Y.-H. Lu, W. Liu, and A. C. Berg. "Low-power image recognition challenge". In: *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE. 2017, pp. 99–104.

[4] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, and V. Boeva. "Hoeffding Trees with nmin adaptation". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2018, pp. 70–79.

[5] G. Hulten, L. Spencer, and P. Domingos. “Mining time-changing data streams”. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 97–106.

[6] W. Hoeffding. “Probability inequalities for sums of bounded random variables”. In: *Journal of the American statistical association* 58.301 (1963), pp. 13–30.

[7] M. Dubois, M. Annavaram, and P. Stenström. *Parallel computer organization and design*. Cambridge University Press, 2012.

[8] C. Reams. “Modelling energy efficiency for computation”. PhD thesis. University of Cambridge, 2012.

[9] S. Ruth. “Green it more than a three percent solution?” In: *IEEE Internet Computing* 13.4 (2009).

[10] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner. *United States data center energy usage report*. Tech. rep. Lawrence Berkeley National Laboratory, Berkeley, California, 2016.

[11] B. Rhoden, K. Klues, D. Zhu, and E. Brewer. “Improving per-node efficiency in the datacenter with new OS abstractions”. In: *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 2011, p. 25.

[12] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. “Energy efficiency for large-scale mapreduce workloads with significant interactive analysis”. In: *Proceedings 7th European Conference on Computer Systems*. 2012, pp. 43–56.

[13] R. Evans and J. Gao. *DeepMind Reduces Google Data Centre Cooling Bill by 40%*. Retrieved from `https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/`. Online; accessed 1 December 2019. 2016.

[14] M. M. Gaber, S. Krishnaswamy, and A. Zaslavsky. “On-board mining of data streams in sensor networks”. In: *Advanced methods for knowledge discovery from complex data*. Springer, 2005, pp. 307–335.

[15] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. "Towards an Adaptive Approach for Mining Data Streams in Resource Constrained Environments". In: *"Data Warehousing and Knowl. Discovery: 6th International Conference, Zaragoza, Spain, Sept. 1-3."* Ed. by Y. Kambayashi, M. Mohania, and W. Wöß. Springer, 2004, pp. 189–198.

[16] C. R. center CRC 876 at Technical University of Dortmund. *Resource-Aware Machine Learning*. Retrieved from `http://sfb876.tu-dortmund.de/SPP/index.html`. Online; accesed 1 December 2019. 2017.

[17] I. Korb, H. Kotthaus, and P. Marwedel. "mmapcopy: Efficient Memory Footprint Reduction using Application-Knowledge". In: *SAC 2016 31st ACM Symposium on Applied Computing*. 2016, pp. 1832–1837.

[18] T.-J. Yang, Y.-H. Chen, and V. Sze. "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning". In: *arXiv preprint arXiv:1611.05128* (2016).

[19] T. Meng and B. Yuan. "Parallel edge-based visual assessment of cluster tendency on GPU". In: *International Journal of Data Science and Analytics* 6.4 (Dec. 2018), pp. 287–295. ISSN: 2364-4168. DOI: `10.1007/s41060-018-0100-7`. URL: `https://doi.org/10.1007/s41060-018-0100-7`.

[20] V. Losing, B. Hammer, and H. Wersing. "KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift". In: *IEEE 16th International Conference on Data Mining (ICDM)*. Dec. 2016, pp. 291–300.

[21] N. Kourtellis, G. D. F. Morales, A. Bifet, and A. Murdopo. "VHT: Vertical Hoeffding Tree". In: *arXiv preprint arXiv:1607.08325* (2016).

[22] F. Carcillo, Y.-A. Le Borgne, O. Caelen, and G. Bontempi. "Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization". In: *International Journal of Data Science and Analytics* 5.4 (June 2018), pp. 285–300. ISSN: 2364-4168. DOI: `10.1007/s41060-018-0116-z`. URL: `https://doi.org/10.1007/s41060-018-0116-z`.

[23] N. Moniz, P. Branco, and L. Torgo. "Resampling strategies for imbalanced time series forecasting". In: *International Journal of Data Science and Analytics* 3.3 (May 2017), pp. 161–181. ISSN: 2364-4168. DOI: `10.1007/s41060-017-0044-3`. URL: `https://doi.org/10.1007/s41060-017-0044-3`.

[24] D. Marrón, E. Ayguadé, J. R. Herrero, J. Read, and A. Bifet. "Low-latency multi-threaded ensemble learning for dynamic big data streams". In: *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE. 2017, pp. 223–232.

[25] E. Garcia-Martin, N. Lavesson, and H. Grahn. "Identification of energy hotspots: A case study of the very fast decision tree". In: *International Conference on Green, Pervasive, and Cloud Computing*. Springer. 2017, pp. 267–281.

[26] V. G. T. da Costa, A. C. P. de Leon Ferreira de Carvalho, and S. B. Junior. "Strict Very Fast Decision Tree: A memory conservative algorithm for data stream mining". In: *Pattern Recognition Letters* 116 (2018), pp. 22–28. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2018.09.004.

[27] A. Bifet and R. Gavaldà. "Adaptive learning from evolving data streams". In: *International Symposium on Intelligent Data Analysis*. Springer. 2009, pp. 249–260.

[28] A. Mazouz, D. C. W. 0001, D. J. Kuck, and W. Jalby. "An Incremental Methodology for Energy Measurement and Modeling." In: *ICPE* (2017), pp. 15–26.

[29] J. G. Koomey, S. Berard, M. Sanchez, and H. Wong. "Assessing trends in the electrical efficiency of computation over time". In: *IEEE Annals of the History of Computing* 17 (2009).

[30] N. Weste, D. Harris, and A. Banerjee. "Cmos vlsi design". In: *A circuits and systems perspective* 11 (2005), p. 739.

[31] M. Horowitz. "1.1 computing's energy problem (and what we can do about it)". In: *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*. IEEE. 2014, pp. 10–14.

[32] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. "RAPL: Memory power estimation and capping". In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. Aug. 2010, pp. 189–194. DOI: 10.1145/1840845.1840883.

[33] A. Bifet, J. Zhang, W. Fan, C. He, J. Zhang, J. Qian, G. Holmes, and B. Pfahringer. "Extremely Fast Decision Tree Mining for Evolving Data Streams". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2017, pp. 1733–1742.

[34] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. "MOA: Massive Online Analysis". In: *Journal of Machine Learning Research* 11 (2010), pp. 1601–1604. URL: http://portal.acm.org/citation.cfm?id=1859903.

[35] L. Breiman. *Classification and regression trees*. Routledge, 2017.

[36] M. Harries. *Splice-2 comparative evaluation: Electricity pricing*. Tech. rep. 1999.

[37] E. Ikonomovska. *Datasets*. Retrieved from http://kt.ijs.si/elena_ikonomovska/data.html. Online; accessed 1 December 2019. 2013.

[38] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan. "Cost-based modeling for fraud and intrusion detection: Results from the JAM project". In: *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*. Vol. 2. IEEE. 2000, pp. 130–144.

[39] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization." In: *ICISSP*. 2018, pp. 108–116.

[40] Student. "The probable error of a mean". In: *Biometrika* (1908), pp. 1–25.

[41] D. J. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

[42] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, and V. Boeva. "How to Measure Energy Consumption in Machine Learning Algorithms". In: *ECML PKDD 2018 Workshops*. Ed. by C. Alzate, A. Monreale, H. Assem, A. Bifet, T. S. Buda, B. Caglayan, B. Drury, E. García-Martín, R. Gavaldà, I. Koprinska, S. Kramer, N. Lavesson, M. Madden, I. Molloy, M.-I. Nicolae, and M. Sinn. Cham: Springer International Publishing, 2019, pp. 243–255. ISBN: 978-3-030-13453-2.

10

Energy Modeling of Hoeffding Tree Ensembles

Eva García-Martín, Albert Bifet, Niklas Lavesson

Abstract

Energy consumption reduction has been an increasing trend in machine learning over the past few years due to its socio-ecological importance. In new challenging areas such as edge computing, energy consumption and predictive accuracy are key variables during algorithm design and implementation. State-of-the-art ensemble stream mining algorithms are able to create highly accurate predictions at a substantial energy cost. This paper introduces the *nmin adaptation* method to ensembles of Hoeffding tree algorithms, to further reduce their energy consumption without sacrificing accuracy. We also present extensive theoretical energy models of such algorithms, detailing their energy patterns and how *nmin adaptation* affects their energy consumption. We have evaluated the energy efficiency and accuracy of the *nmin adaptation* method on five different ensembles of Hoeffding trees under 11 publicly available datasets. The results show that we are able to reduce the energy consumption significantly, by 21 % on average, affecting accuracy by less than one percent on average.

10.1 Introduction

Energy consumption in machine learning is starting to gain importance in state-of-the-art research. This is clearly visible in areas where researchers are incorporating the inference or training of the model inside the device (i.e. edge computing or edge AI). An area whose main focus is on real-time prediction on the edge is data stream mining.

One of the most well-known and used classifier is the decision tree, due to its explainability advantage. In the data stream setting, where we can

only do one pass over the data, and we can not store all of it, the main problem of building a decision tree is the need of reusing the examples to compute the best splitting attributes. Hulten and Domingos [1] proposed the Hoeffding Tree or VFDT, a very fast decision tree for streaming data, where instead of reusing instances, we wait for new instances to arrive. The most interesting feature of the Hoeffding tree is that it builds an identical tree with a traditional one, with high probability if the number of instances is large enough, and that it has theoretical guarantees about that.

Decision trees are usually not used alone, but within ensembles methods. Ensemble methods are combinations of several models whose individual predictions are combined in some manner (e.g., averaging or voting) to form a final prediction. They have several advantages over single classifier methods: they are easy to scale and parallelize, they can adapt to change quickly by pruning under-performing parts of the ensemble, and they therefore usually also generate more accurate concept descriptions.

Advancements in data stream mining have been primarily focused on creating algorithms that output higher predictive performance. For that, they used ensembles of existing algorithms. However, these solutions output high predictive performance at the cost of higher energy consumption. To address this challenge, we present the *nmin adaptation* method for ensembles of Hoeffding Tree algorithms [1], to make them more feasible to run in the edge.

The *nmin adaptation* method is a method presented in [2], which reduces the energy consumption of standard Hoeffding tree algorithms by adapting the number of instances needed to create a split. We extend that study by incorporating the *nmin adaptation* method to ensembles of Hoeffding Trees. The goal of this paper is two-fold:

- Present an energy efficient approach to real-time prediction with high levels of accuracy.
- Present detailed theoretical energy models for ensemble of Hoeffding trees, together with a generic approach to create energy models, applicable to any class of algorithms.

We have conducted experiments on five different ensembles of Hoeffding trees (Leveraging Bagging [3], Online Coordinate Boosting [4], Online Accuracy Updated Ensemble [5], Online Bagging [6],and Online Boosting [6]),

with and without *nmin adaptation*, on 11 publicly available datasets. The results show that we are able to reduce the energy consumption by 21%, affecting accuracy by less than 1%, on average.

This approach achieves similar levels of accuracy as state-of-the-art ensemble online algorithms, while significantly reducing its energy consumption. We believe this is a significant step towards a greener data stream mining, by proposing not only more energy efficient algorithms, but also creating a better understanding of how ensemble online algorithms consume energy.

The rest of the paper is organized as follows: The work related to this study is presented in Section 10.2. Background on Hoeffding trees and the *nmin adaptation* method is presented in Section 10.3. The main contribution of this paper is presented in Section 10.4. There we first describe how to build energy models for any kind of algorithm (Fig. 10.2), we then present extensive theoretical energy models for the Online Bagging, Leveraging Bagging, Online Boosting, Online Coordinate Boosting, and Online Accuracy Updated Ensemble algorithms. This helps at understanding the energy bottlenecks of the algorithm, and why *nmin adaptation* is efficient at handling those bottlenecks. The design of the experiments together with the results are presented in Sections 10.5 and 10.6 respectively. The paper ends with conclusions in Section 10.7.

10.2 Related Work

Energy efficiency has been widely studied in the field of computer architecture for decades [7]. Moore's law stated that the performance of CPUs was going to double every 18 months, by doubling the number of transistors. This prediction was made considering one fundamental constraint, that the energy consumed by each unit of computing would decrease as the number of transistors increased[1]. The CPU scaling from Moore does not apply anymore, and the main reason is because there was not enough focus on scaling the power while increasing performance [8]. This has created a paradigm shift, where power is the key factor studied to improve computer performance, creating processors that handle more operations using the same amount of power [9].

[1] `https://www.forbes.com/2010/04/29/moores-law-computing-processing-opinions-contributors-bill-dally.html`

Machine learning research, and in particular deep learning, is aware of the need to focus not only on creating more accurate models, but also on creating energy efficient models [10–12]. The researchers in this area have realized that the amount of energy and resources (e.g. GPUs) needed to perform training and inference on such models is impractical, especially for mobile platforms. Currently most of the machine learning models used in the phone are accessed through the cloud, since doing inference in the phone, which improves user privacy, is simply not possible at the moment. Some research is going in that direction, such as *The Low Power Image Recognition Challenge* (LPIRC) [13], and the work by [14, 15].

Approaches to mine streams of data have been evolving during the past years. We consider the Hoeffding tree algorithm [1] to be the first algorithm that was able to classify data from an infinite stream of data in constant time per example. That approach was later improved to be able to handle concept drift [16], that is, non-stationary streams of data that evolve through time. The ability to handle concept drift introduces many computational demands, since the algorithm needs to keep track of the error to detect when a change occurs. ADWIN (adaptive windowing) [17] is the most efficient change detector that can be incorporated to any algorithm. The Hoeffding Adaptive Tree (HAT) [18] was introduced as an extension to the standard Hoeffding tree algorithm that is able to handle concept drift using the ADWIN detector.

In relation to energy efficiency and data stream mining, several studies have recently focused on creating more energy efficient versions of existing models. The Vertical Hoeffding Tree (VHT) [19] was introduced to parallelize the induction of Hoeffding trees. Another approach for distributed systems is the Streaming Parallel Decision Tree algorithm (SPDT) [20]. Marrón et al. [21] propose a hardware approach to improve Hoeffding trees, by parallelizing the execution of random forests of Hoeffding trees and creating specific hardware configurations. Another streaming algorithm that was improved in terms of energy efficiency was the KNN version with self-adjusting memory [22].

A recent publication at the International Conference on Data Mining [23] presented an approach similar to ours, which estimates the value of *nmin* to avoid unnecessary split attempts. Although their approach seems to better estimate the value of *nmin*, our approach is still more energy efficient (based on their results), which is the ultimate goal of our study. We believe that

trading off a few percentages of accuracy is necessary in some scenarios (e.g. embedded devices) where energy and battery consumption is the main concern. This work is an extension of an already published study where we introduced *nmin adaptation* [2]. While in that work the *nmin adaptation* method was only applied to the standard version of the Hoeffding tree algorithm, this study proposes more energy efficient approaches to create ensembles of Hoeffding trees, validated by the experiments on five different algorithms and 11 different datasets.

10.3 Background

This section explains in detail Hoeffding trees, together with the ensembles of Hoeffding trees used in this study.

10.3.1 Hoeffding Tree algorithm

The Hoeffding tree algorithm (Alg. 8), also known as Very Fast Decision Tree (VFDT) [1], was first introduced in the year 2000, presenting the first approach that was able to mine from a stream of data in constant time per example, with low computational constraints, and being able to read the data only once without storing it. The algorithm builds and updates the model as the data arrive, storing only the necessary statistics at each node to be able to grow the tree.

The algorithm reads an instance, traverses the tree until it reaches the corresponding leaf, and updates the statistics at that leaf based on the information from that instance. For attributes with discrete values, the statistics are counts of class values for each attribute value. On the other hand, for attributes with numerical values there are several approaches to safe the information in the most efficient way. The most common approach used nowadays is to maintain a Gaussian function with the mean, standard deviation, maximum, and minimum, at each node, for each class label and attribute.

Once *nmin* instances are read in a particular leaf, the algorithm calculates the information gain(entropy) for each attribute, using the aforementioned statistics. If $\Delta\overline{G} > \epsilon$, i.e. the difference between the information gain from the best and the second best attribute ($\Delta\overline{G}$) is higher than the Hoeffidng Bound[24](ϵ), then a split occurs, and the leaf substituted by the node with

the best attribute. The Hoeffding bound(ϵ) is defined as:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{10.1}$$

and it states that the split on the best attribute having observed n number of examples, will be the same as if the algorithm had observed an infinite number of examples, with probability $1 - \delta$. The idea is that the Hoeffding tree can be approximated to a standard decision tree by observing a sufficient number of instances at each node, to create a confident split. On the other hand, if $\Delta\overline{G} < \epsilon < \tau$, a tie occurs. This happens when the two top attributes are equally good, thus the tree splits on any of those.

Algorithm 7 AttemptSplit **Symbols:** X: attributes; ϵ:Hoeffding bound(Eq.(10.1))

1: Compute $\overline{G_l}(X_i)$ for each attribute X_i
2: $\Delta\overline{G} = \overline{G}_l(X_a) - \overline{G}_l(X_b)$\{Difference between two best attributes\}
3: **if** $(\Delta\overline{G} > \epsilon)$ or $(\epsilon < \tau)$ **then**
4: Split $\leftarrow$ True
5: **end if**

10.3.2 *nmin* adaptation

The *nmin adaptation* method was previously introduced in [2]. While that study only focused on the energy reduction of standard Hoeffding tree algorithms, this study proposes to use the *nmin adaptation* method on ensembles of Hoeffding trees, to align with the current approaches in data stream mining.

The goal of the *nmin adaptation* method is to estimate the number of instances (*nmin*) that are needed to make a split with confidence $1 - \delta$. Current Hoeffding tree algorithms use a fixed value of *nmin* instances. Thus the batch of instances that are observed on each node before checking for a possible split is the same for each node, and during the complete execution of the algorithm. As is explained in Section 10.4.2, having a fixed *nmin* value is energy inefficient. This is because there are many times where those instances are not enough to make a confident split, thus all those computations are done unnecessarily.

Our method estimates, for each node, the *nmin* instances that ensure a split, to calculate the splitting attributes only when there is going to be a

Algorithm 8 Hoeffding Tree. **Symbols:** HT: Initial tree; X: set of attributes; $G(\cdot)$: split evaluation function; τ: hyperparameter used for attributes with tied information gain; $nmin$: hyperparameter to decide when to check for a split.

while stream is not empty **do**
  Read instance
  Traverse the tree using HT {Until leaf l is reached}
  Update statistics at leaf l {Nominal and numerical attributes}
  Increment n_l: instances seen at l
  if $nmin \leq n_l$ **then**
    AttemptSplit(l){Call Function from Alg. 7}
    if Split==True **then**
      CreateChildren(l){New leaf l_m with initialized statistics}
    else
      Disable attr $\{X_p | (\overline{G}_l(X_p) - \overline{G}_l(X_a)) > \epsilon\}$
    end if
  end if
end while

split. Since each node will have their own $nmin$, we allow the tree to grow faster (lower $nmin$) in those branches where there is a clear attribute to split on, and to delay the growth (higher $nmin$) in those branches where there is no clear attribute to split on.

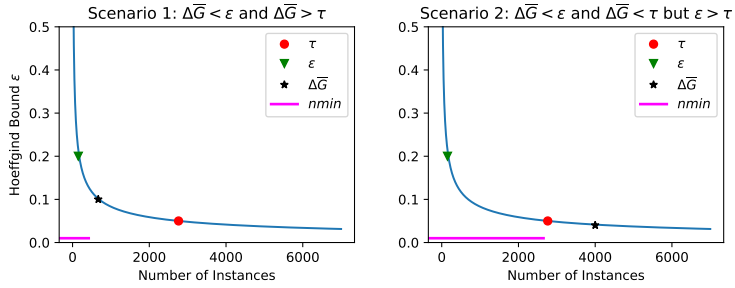

Figure 10.1: *Example of the nmin adaptation method. The value of nmin is going to be adapted based on two scenarios: scenario 1 (left plot), and scenario 2 (right plot)*

The value of $nmin$ is going to be adapted only when a non-split occurs, otherwise we assume that the current value is the optimal one. The method

follows two approaches, illustrated in Figure 10.1.

The first approach (scenario 1) approximates the value of *nmin* when the non-split occurred because no attribute was the clear winner (Figure 10.1 left plot). In this case, *nmin* is estimated as the number of instances needed for the best attribute to be higher than the second best attribute with a difference of ϵ. Taking a look at the left plot from Figure 10.1, *nmin* are the instances needed for the green triangle (ϵ), to reach the black star($\Delta\overline{G}$). When we wait for those *nmin* instances, then $\Delta\overline{G} > \epsilon$, satisfying the condition for a split. More formally, for the first scenario:

$$nmin = \left\lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot (\Delta G)^2} \right\rceil \tag{10.2}$$

The second approach (scenario 2) approximates the value of *nmin* when the non-split occurred because $\epsilon > \tau$. In this scenario, although the attributes are very similar ($\Delta\overline{G} < \epsilon$), and their difference in entropy is smaller than τ, the confidence is still not high enough to make a split. Taking a look at the right plot from Figure 10.1, *nmin* are the instances needed for the green triangle (ϵ), to reach the red dot(τ). When we wait for those instances, then $\epsilon < \tau$, satisfying the condition to create a split. More formally, the *nmin* approximation for the scenario 2 is defined as:

$$nmin = \left\lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot \tau^2} \right\rceil \tag{10.3}$$

The *nmin adaptation* implemented for the Hoeffding tree algorithm is portrayed in Alg. 9.

10.4 Energy Modeling of Hoeffding Tree Ensembles

This section presents the main contribution of this paper, guidelines on how to construct energy efficient ensembles of Hoeffding trees, validated with the experiments on Section 10.6. In order to reduce the energy consumption of ensembles of Hoeffding trees, we apply the nmin adaptation method to the Hoeffding tree algorithm, and use that algorithm as the base for the different ensembles. To have a more detailed view on how energy is consumed on ensembles of Hoeffding tees, we first portray an approach to create generic theoretical energy models for any kind of algorithm (Section 10.4.1). We then show how that generic approach applied for ensemble of Hoeffding trees

Algorithm 9 Hoeffding Tree with nmin adaptation. **Symbols:***nmin*: hyperparameter initially set by the user that is going to be adapted; HT: Initial tree; X: set of attributes; $G(\cdot)$: split evaluation function; τ: hyperparameter used for attributes with tied information gain

while stream is not empty **do**
  Read instance
  Traverse the tree using HT {Until leaf l is reached}
  Update statistics at leaf l {Nominal and numerical attributes}
  Increment n_l: instances seen at l
  if $nmin \leq n_l$ **then**
    AttemptSplit(l){Call Function from Alg. 7}
    if Split==True **then**
      CreateChildren(l){New leaf l_m with initialized statistics}
    else
      Disable attr $\{X_p|(\overline{G}_l(X_p) - \overline{G}_l(X_a)) > \epsilon\}$ {Adapt the value of nmin}
      if $\Delta\overline{G} \leq \tau$ **then**
        $nmin = \left\lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot \tau^2} \right\rceil$ {Scenario 2}
      else
        $nmin = \left\lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot (\Delta G)^2} \right\rceil$ {Scenario 1}
      end if
    end if
  end if
end while

(Section 10.4.2 and Fig. 10.3). That section finalizes with a detailed energy model and explanation of each algorithm: Online Bagging, Online Boosting, Leveraging Bagging, Online Coordinate Boosting, Online Accuracy Updated Ensemble, and Hoeffding tree with *nmin adaptation*.

10.4.1 Generic approach to create energy models

We have summarized the different steps to create theoretical energy models in Figure 10.2.

First, *Step 1* formalizes the generic energy model, as a sum of the different operations (mapped then to algorithm functions) categorized by type of operation. Thus, the total energy of a model can be expressed as:

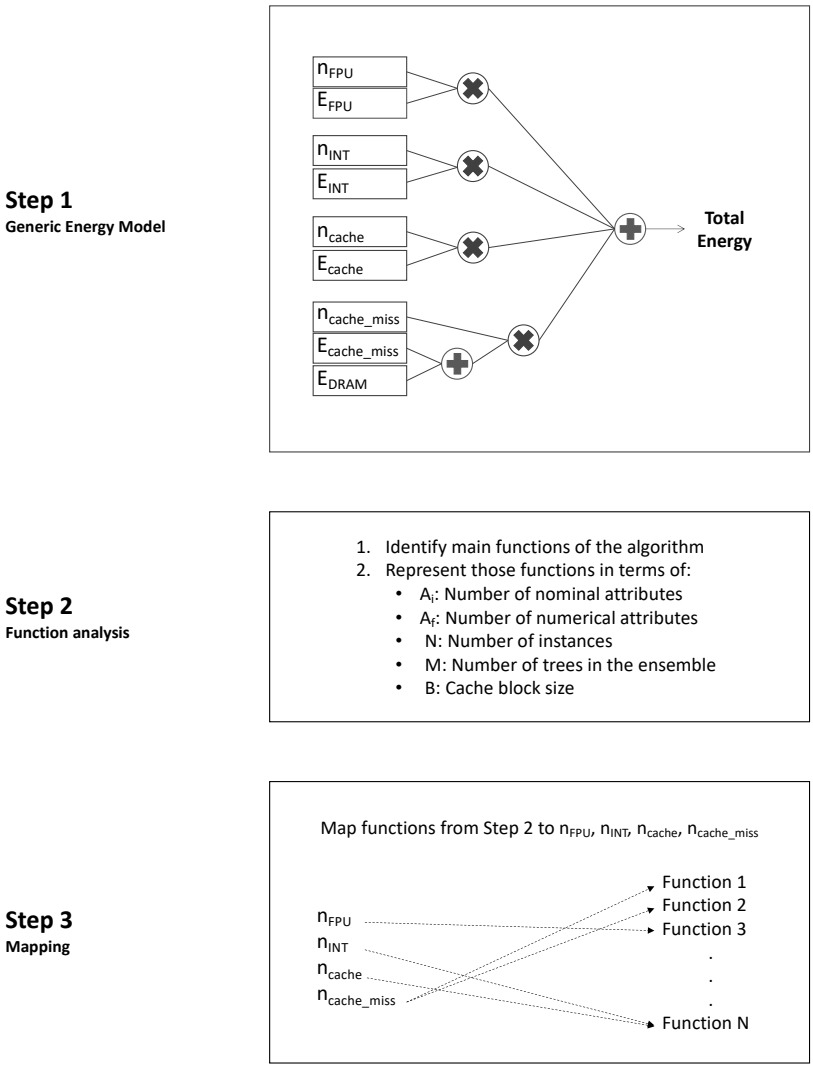

Figure 10.2: *Generic approach to build energy models for any kind of algorithm*

$$\begin{aligned} E = & \, n_{FPU} \cdot E_{FPU} \\ & + \, n_{INT} \cdot E_{INT} \\ & + \, n_{cache} \cdot E_{cache} \\ & + \, n_{cache\_miss} \cdot (E_{cache\_miss} + E_{DRAM}) \end{aligned} \tag{10.4}$$

The operations are divided into integer, floating point operations, and DRAM and cache accesses. We consider a DRAM access everytime a cache miss occurs. The energy per access or per computation (E_{FPU}, E_{INT}, E_{cache_miss}, E_{DRAM}) will depend on the specific hardware where the algorithm is run. However, knowing which type of operations consumes more energy (example a DRAM access consumes 100 times more energy than a floating point operation), together with the amount of operations for each type, gives a detailed overview of the theoretical energy consumption of the algorithm. *Step 1* can be done only once, as appears in the figure, and can be used for any algorithm. It can be adapted to specific hardware components if needed.

Step 2 focuses on identifying the main functions of the algorithm, and then representing them in terms of generic features such as number of attributes or size of the ensemble.

In *Step 3* the goal is to map the functions from *Step 2* to the different type of operations. For example, a function that counts the number of instances can be expressed as one integer operation per instance.

An example of how to apply these steps is shown in Section 10.4.2.

10.4.2 Ensembles Energy Models

The process to create an energy model for any ensemble of other algorithms is summarized in Fig. 10.3. The goal is to combine the energy model of the ensemble and the energy model of the base algorithm for the ensemble. In the case of Online Bagging, for instance, we need to sum the energy model of Online Bagging plus the energy model of the Hoeffding tree with *nmin adaptation*. If another model is used in the ensemble, it can be substituted by the Hoeffding tree with nmin adaptation (dashed box).

This section details, first, the energy model of Hoeffding Trees with *nmin adaptation* (Fig. 10.4), then the energy models of the ensembles: Online Bagging (Fig. 10.5), Leveraging Bagging (Fig. 10.6), Online Boosting (Fig. 10.7), Online Coordinate Boosting (Fig. 10.8), Online Accuracy Updated Ensemble (Fig. 10.9).

Hoeffding Trees with nmin adaptation The energy consumption of Hoeffding tress with *nmin adaptation* (HT-*nmin* for the remainder of the paper), explained in detail in Section 10.3.2, is summarized with the energy model from Fig. 10.4

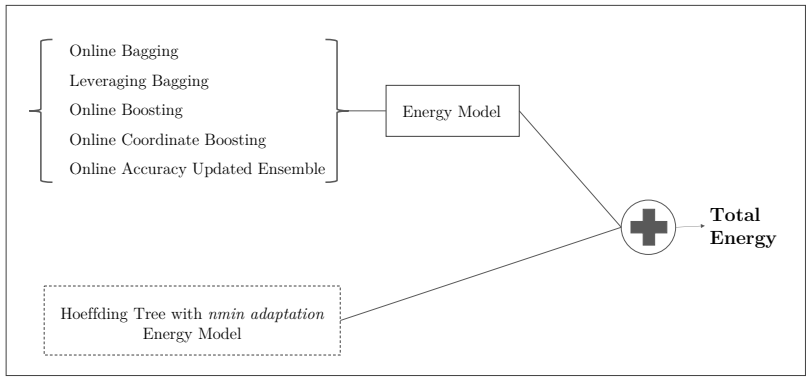

Figure 10.3: *Energy model approach for ensembles of Hoeffding trees with nmin adaptation. Steps: i) Obtain the energy model (following Fig. 10.2) for the general ensemble of trees; ii) Obtain the energy model of Hoeffding trees with nmin adaptation; iii) Sum the energy model variables. The energy model for Hoeffding trees (dashed box) can be substituted by the energy model of any other algorithm that is going to be part of the ensembles (for example Hoeffding Adaptive Trees [18])*

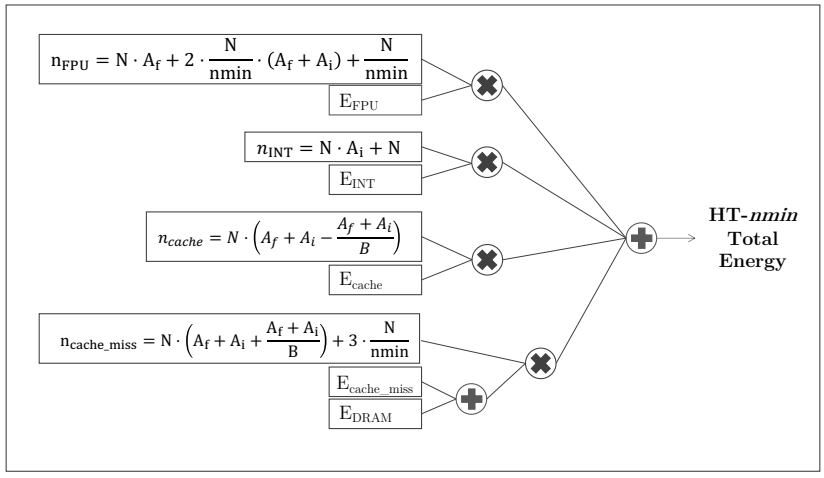

Figure 10.4: *Energy model for the Hoeffding tree with nmin adaptation*

The second step (Fig. 10.2) after defining the generic energy model is to identify the main functions of the HT-*nmin*. The HT-*nmin* algorithm spends energy on training (traversing the tree, updating the statistics, checking for a split, doing a split), and doing inference (traversing the tree, Naive Bayes

or majority class prediction at the leaf). In particular:

- **Traverse the tree**: The energy spent on traversing the tree is calculated as the number of cache misses, which we assume as one access per attribute per instance (for every time as instance is read), as the worst case scenario.

- **Updating statistics**: The energy spent on updating statistics is divided in nominal and numerical attributes. The energy spent in updating attributes regarding computations is one integer computation per instance per nominal attribute, and one floating point operation per numerical attribute. In terms of accesses, for both nominal and numeric attributes, we have a cache miss the first time we access the table, and then one cache hit per nominal attribute, and then a cache miss for every attribute that exceeds the block size.

- **Checking for a split**: To check for a split we need to calculate the entropy for all attributes, calculate the Hoeffding bound, and sort the attributes to obtain the best one. Calculating the entropy, calculating the Hoeffding bound, and sorting the attributes, is one floating point operation per attribute, every *nmin* instances. Regarding memory accesses, we consider one cache miss every *nmin* instances.

- **Doing a split**: To create a new node we consider one floating point operation and one cache miss every *nmin* instances.

Finally, those functions are mapped to the corresponding n_{FPU}, n_{INT}, n_{cache} , n_{cache_miss}, as shown in Fig. 10.4.

What we believe is more important to observe from this energy model is the impact of the *nmin* parameter. The amount of times that the algorithm check for split, which involves a significant part of the energy consumption, depends on the value of *nmin*. That is why having a bad approximation of *nmin* incurs in high energy costs without increasing the predictive performance. This model is the input presented in the energy models of the ensembles from the following paragraphs. .

Online Bagging Online Bagging, presented in Algorithm 10, was proposed by Oza and Russell [6] as a streaming version of traditional ensemble bagging.

Bagging is one of the simplest ensemble methods to implement. Non-streaming bagging [25] builds a set of M base models, training each model with a bootstrap sample of size N created by drawing random samples with replacement from the original training set. Each base model's training set contains each of the original training examples K times where $P(K = k)$ follows a binomial distribution:

$$P(K = k) = \binom{n}{k} p^k (1-p)^{n-k} = \binom{n}{k} \frac{1}{n}^k \left(1 - \frac{1}{n}\right)^{n-k}$$

This binomial distribution for large values of n tends to a Poisson(1) distribution, where Poisson(1)$= \exp(-1)/k!$. Using this fact, [6] proposed *Online Bagging*, an online method that instead of sampling with replacement, gives each example a weight according to Poisson(1).

Algorithm 10 Online Bagging(h,d)

```
1: for each instance d do
2:    for each model h_m, (m ∈ 1, 2, ..., M) do
3:       k = Poisson(λ = 1)
4:       Train the model on the new instance k times
5:    end for
6: end for
```

This algorithm has two main functions, one involved in training the model, that depends on the base algorithm, and the other function that calculates k. Training the model can be represented as the energy model of the Hoeffding Tree with *nmin adaptation*, from Fig. 10.4. Calculating k involves one floating point operation per instance (N) per model(M). The final energy model for Online Bagging is represented in Fig. 10.5.

Leveraging Bagging When data is evolving over time, it is important that models adapt to the changes in the stream and evolve over time. `ADWIN` bagging [26] is the online bagging method of Oza and Russell with the addition of the `ADWIN` algorithm [17] as a change detector and as an estimator for the weights of the boosting method. When a change is detected, the worst classifier of the ensemble of classifiers is removed and a new classifier is added to the ensemble. *Leveraging Bagging* [3], shown in Alg. 12, extends `ADWIN` bagging by leveraging the performance of bagging with two randomization improvements: increasing resampling and using output detection codes.

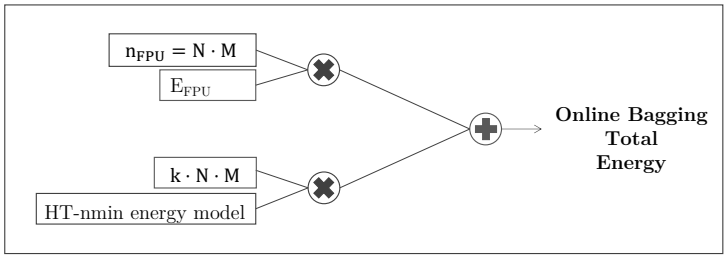

Figure 10.5: *Online Bagging energy model based on the number of models (M) and number of instances (N), considering the Hoeffding Tree with nmin adaptation as the base tree classifier.*

Leveraging bagging increases the weights of this resampling using a larger value λ to compute the value of the Poisson distribution. For every instance, on top of training the model for each ensemble as for Online Bagging, Leveraging Bagging checks if the instance is correctly classified by the model, and inputs such error value to the `ADWIN` detector (Alg. 11), to check for a possible change.

Algorithm 11 `ADWIN`. **Symbols:** h: ensemble of models h_m: model in particular. d: instance

```
1: y_pred = Classify instance d with classifier h_m
2: y = True class of d
3: if y_pred == y then
4:    correctlyClassifies ← True
5: end if
6: ErrEstim = getEstimation(h_m)
7: if setInput(h_m, correctlyClassifies) == True then
8:    if getEstimation(h_m) > ErrEstim then
9:       Change ← True
10:   end if
11: end if
```

Leveraging Bagging has a similar energy consumption pattern than Online Bagging with the addition of the `ADWIN` detector and the output codes. The `ADWIN` detector introduces significant overhead since it has to keep track of the error. For every instance `ADWIN` calculates if model h_m correctly classifies instance d, for each model $h_m, (m \in 1, 2, ..., M)$. It then

Algorithm 12 Leveraging Bagging(h,d,λ)

```
1: for each instance d do
2:   for each model h_m, (m ∈ 1, 2, ..., M) do
3:     k = Poisson(λ_d)
4:     Train the model on the new instance k times
5:     ADWIN (h, h_m, d){Call function from Alg. 11}
6:     if ADWIN detects change on h_m then
7:       Replace classifier with highest error with a new classifier
8:     end if
9:   end for
10: end for
```

inputs a 0 for misclassification or 1 for a correct classification to the `ADWIN` detector. With this information, `ADWIN` outputs if there has been a change in the current window. If the size of the window exceeds the maximum value assigned, `ADWIN` resizes the window. Finally, if a change is detected, the worst performing classifier is removed from the ensemble, and a new one is created.

The main functions of the baseline algorithm are: calculate k, train with HT-nmin, and replace classifier. The main functions of `ADWIN` are: traverse the tree, obtain the true class y, compare y with the prediction, get the estimation, and set the input to `ADWIN`. The following list details these functions in terms of type of operations and number of instances, models, etc. Computing the output codes are omitted since they are not activated by default in the algorithm.

- **Calculate** k is one FPU operation per instance per model.
- **Replacing the classifier** is one cache miss per classifier to get the error of each classifier, and one cache miss to replace it. That is per instance per model per everytime change is detected (S variable).
- **Traverse the tree**. As for the HT-nmin, the energy is calculated as the number of cache misses, assuming one access per attribute per instance.
- **Obtaining the true class** is one cache miss per instance per model
- **Comparing** y **with the prediction** is one cache access per instance per model.

- **Getting the estimation** is just dividing the amount of error by the number of instances, which outputs to one floating point operation per instance per model.

- **Setting the input** is quite energy consuming, since there are several operations involved. First it inserts the element in the window bucket, which is one cache miss per instance per model. Second it compresses the buckets, which we calculate as one cache miss per instance per model for the first access, and then $(log(W) - 1)$ cache accesses per instance per model for the rest of the window accesses, W being the window size. Third it reduces the window, which is the same energy consumption pattern as compressing the buckets, one cache miss per instance per model and $(log(W) - 1)$ cache accesses per instance per model.

Figure 10.6 shows the energy model of the Leveraging Bagging algorithm, based on the function explanations presented in the previous list. M is the number of models in the example, N the number of instances, W is the window size for `ADWIN`, s is the speed of change (the percentage of instances with change), c is the percentage of correctly classified instances.

Online Boosting Online Boosting [27, 28] is an extension of boosting for streaming data, presented in Alg. 13. The main difference with Online Bagging is that Online Boosting updates the λ of the Poisson distribution for the next classifier based on the preformance of the current classifier (Fig. 10.4). If classifier m classifies the instance correctly, that instance will be assigned a lower weight for the next classifier $m + 1$. On the other hand, if the instance is classified incorrectly, that instance gets updated with a higher weight. Each instance is passed through each model in sequence.

Regarding its energy consumption and energy model, we observe that Online Boosting presents a similar energy consumption pattern than Online Bagging. Online Bagging introduces an extra memory access to update the weight, which we assume that the access is a DRAM access because each instance is updated sequentially, not fitting in cache. We assume one instance update per instance per model, the worst case scenario if all instances are classified incorrectly. The detailed energy model is presented in Fig. 10.7.

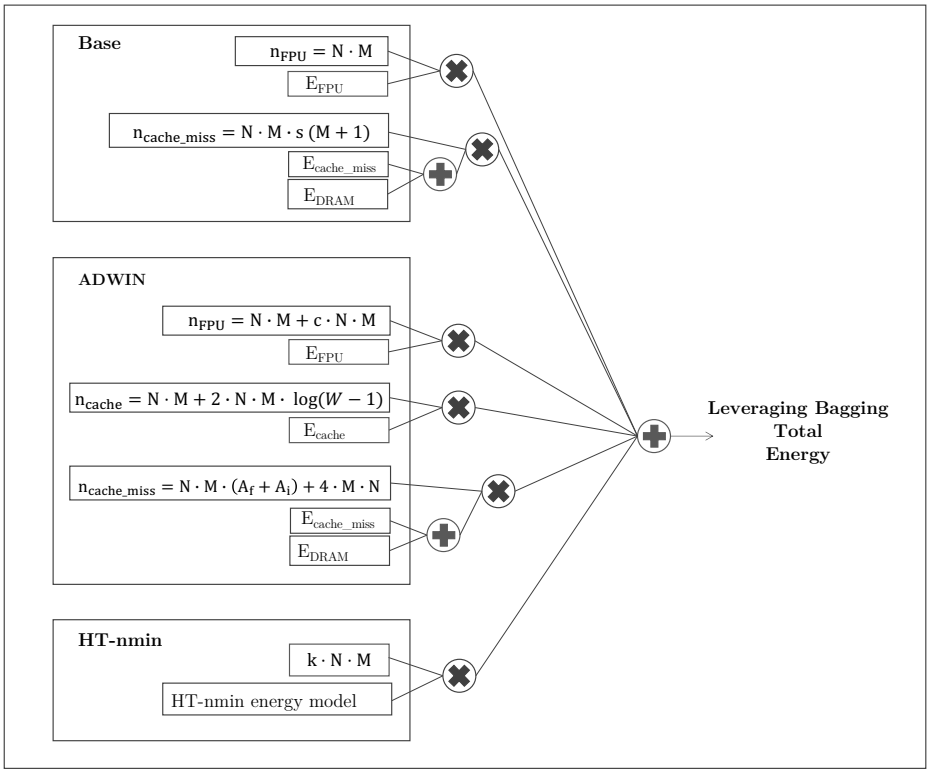

Figure 10.6: *Energy model for Leveraging Bagging. N=number of instances, M=number of models, W=window size, s=speed of change, c=percentage of correctly classified instances*

Online Coordinate Boosting Online Coordinate Boosting [4](Alg. 14) is an extension to Online Boosting, that uses a different procedure to update the weight, to yield a closer approximation to Freund and Schapire's AdaBoost algorithm [29]. The weight update procedure is derived by minimizing AdaBoost's loss when viewed in an incremental form.

From Alg. 14 we can observe the different calculations in lines 8,11,13,14, and 15. Once the weight is calculated, the HT-nmin algorithm is trained with weight d. They use the following formulas to calculate π_j^+, π_j^+, W_{jk}^+ and W_{jk}^-.

Algorithm 13 Online Boosting(h,d)

1: $\lambda_d = 1$
2: **for** each instance d **do**
3: **for** each model $h_m, (m \in 1, 2, ..., M)$ **do**
4: $k = Poisson(\lambda_d)$
5: Train the model on the new instance k times
6: **if** instance is correctly classified **then**
7: instance weight λ_d updated to a lower value
8: instance weight λ_d updated to a higher value
9: **end if**
10: **end for**
11: **end for**

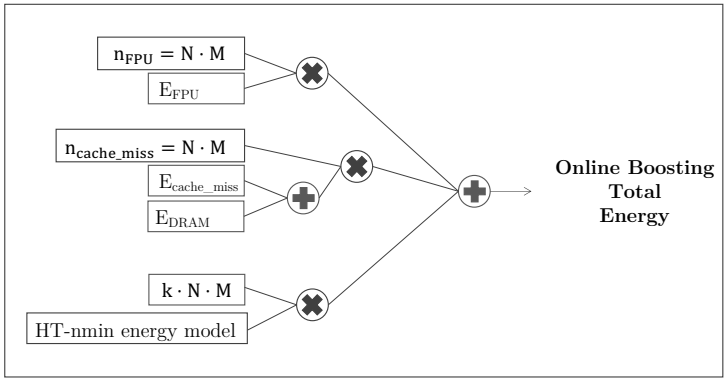

Figure 10.7: *Online Boosting energy model based on the number of models (M) and number of instances (N), considering the Hoeffding Tree with nmin adaptation as the base tree classifier. k represents the output value of the Poisson distribution from Alg. 13, line 4.*

$$\pi_j^+ = \prod_{k=j_0}^{j-1} \left(\frac{W_{jk}^+}{W_{jj}^+} e^{-\Delta\alpha_k} + \left(1 - \frac{W_{jk}^+}{W_{jj}^+}\right) e^{\Delta\alpha_k} \right) \tag{10.5}$$

$$\pi_j^- = \prod_{k=j_0}^{j-1} \left(\frac{W_{jk}^-}{W_{jj}^-} e^{-\Delta\alpha_k} + \left(1 - \frac{W_{jk}^-}{W_{jj}^-}\right) e^{\Delta\alpha_k} \right) \tag{10.6}$$

$$W_{jk}^+ = W_{jk}^+ \pi_j^+ + d\mathbf{1}_{[m_{ik}=+1]} \cdot \mathbf{1}_{[m_{ij}=+1]} \tag{10.7}$$

$$W_{jk}^- = W_{jk}^- \pi_j^- + d\mathbf{1}_{[m_{ik}=-1]} \cdot \mathbf{1}_{[m_{ij}=-1]} \tag{10.8}$$

Algorithm 14 Online Coordinate Boosting(h,x)

1: $d = 1$
2: **for** each instance x **do**
3: **for** each model $h_m, (m \in 1, 2, ..., M)$ **do**
4: **if** instance is correctly classified **then**
5: $m_j = 1$
6: **end if**
7: **for** every model seen so far -1 **do**
8: Calculate π_j^+ and π_j^+ using Eqs. 10.5 and 10.6
9: **end for**
10: **for** every model seen so far **do**
11: Calculate W_{jk}^+ and W_{jk}^- using Eqs. 10.7 and 10.8
12: **end for**
13: $\alpha_j^i = \frac{1}{2} log \frac{W_{jj}^+}{W_{jj}^-}$
14: $\Delta\alpha_j = \alpha_j^i - \alpha_j^{i-1}$
15: $d \leftarrow de^{-\alpha_j^i m_{ij}}$
16: Train the model on the new instance with weight d
17: **end for**
18: **end for**

where j represents the model of the ensemble, j_0 is set to 0, and m_j is defined in line 5 of the algorithm. To iterate over the product we introduced the loops in lines 7 and 10.

In relation to energy consumption, this energy model, presented in Fig. 10.8 adds the extra floating point operations of calculating the values of W_{jk}, π_j, α_j^i, $\Delta\alpha_j$, and d, and the cache accesses related to traversing the tree to calculate if the instance is correctly classified. Traversing the tree is the same as for `ADWIN`, one access per attribute per instance. Calculating π_j can be estimated as

$$\sum_{k=0}^{M-1} k,$$

and W_{jk} as

$$\sum_{k=0}^{M} k.$$

Since the sum of π_j and W_{jk} result in M^2 operations, the amount of floating point operations of lines 8 and 11 can be simplified as: $M \cdot N(M^2 \cdot 2 + 3)$.

Finally, lines 13, 14, and 15 of Alg. 14 require one floating point operation per instance per model.

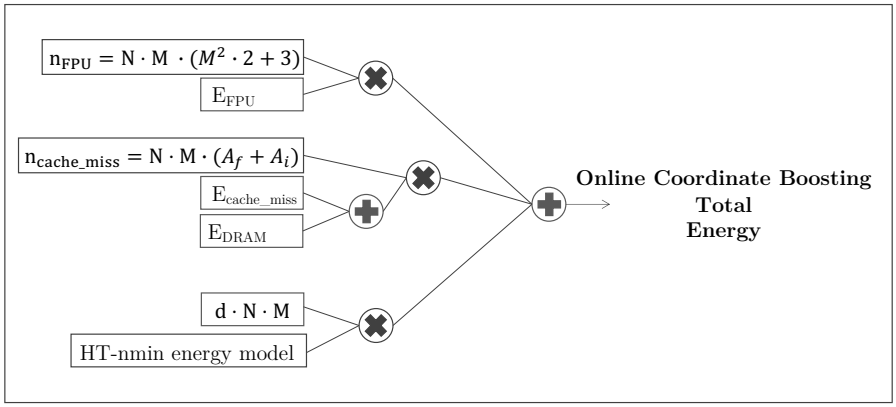

Figure 10.8: *Energy model for Online Coordinate Boosting*

Online Accuracy Updated Ensemble Batch-incremental methods create models from batches, and they remove them as memory fills up; they cannot learn instance-by-instance. The size of the batch must be chosen to provide a balance between best model accuracy (large batches) and best response to new instances (smaller batches). The *Online Accuracy Updated Ensemble* (OAUE) [5, 30], presented in Alg. 15 maintains a weighted set of component classifiers and predicts the class of incoming examples by aggregating the predictions of components using a weighted voting rule. After processing a new example, each component classifier is weighted according to its accuracy and incrementally trained.

Regarding its energy consumption, we first present the main functions of the algorithm, to then present its energy patterns, combined in an energy model on Figure 10.9. The main functions are: create a new classifier, calculate the prediction error, add classifier, weight classifier, and replace classifier.

- **Creating a new classifier** requires one cache miss every w instances (lines 5 and 12), thus $\frac{N}{w}$.
- **Calculating the prediction error** is the same energy as traversing the tree, that as previously explained for the HT-nmin and the Lever-

Algorithm 15 Online Accuracy Updated Ensemble. **Symbols:** ε: ensemble w: window M: ensemble size

```
1: ε ← ∅
2: C′ ←new candidate classifier
3: for each instance x^t do
4:    Calculate the prediction error of all classifiers C_i ∈ ε on x^t
5:    if t > 0 and t mod w = 0 then
6:       if |ε| < M then
7:          Add classifier C′ to the ensemble ε
8:       else
9:          weight all classifiers C_i ∈ ε and C′using Eq. 10.9
10:         substitute least accurate classifier in ε with C′
11:      end if
12:      C′ ← new candidate classifier
13:   else
14:      incrementally train classifier C′ with x_t
15:      weight all classifiers C_i ∈ ε using Eq. 10.9
16:   end if
17:   for all classifiers C_i ∈ ε do
18:      incrementally train classifier C_i with x^t
19:   end for
20: end for
```

aging Bagging algorithms is one cache miss per attribute per instance (line 4), per model.

- **Adding a new classifier** (line 7) is one cache miss every w instances, only the first M times.
- **Calculating the weight** for each classifier requires one floating point operation per classifier per $N - \frac{N}{w}$ instances, using the Eq. 10.9 on line 9, and then on line 15 for the rest of the instances. Thus it requires N floating point operations minus M (condition on line 6).
- **Replacing the classifier** is the same as the function used in `ADWIN`, one cache miss per classifier to calculate the error for each classifier, plus one extra cache miss to replace it. This occurs for every $\frac{N}{w}$ instances.

$$w_i^t = \frac{1}{MSE_r^t + MSE_i^t + \epsilon} \tag{10.9}$$

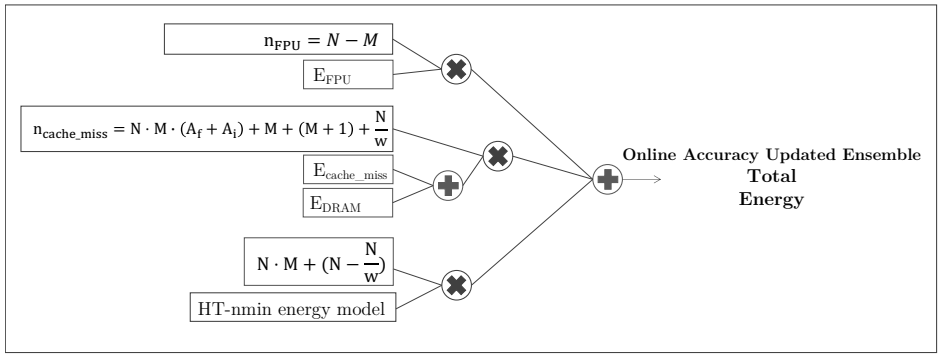

Figure 10.9: *Energy model for Online Accuracy Updated Ensemble*

10.5 Experimental Design

We have performed several experiments to evaluate our proposed method, *nmin adaptation*, in ensembles of Hoeffding Trees. The experiments have a dual purpose: i) first, to see how much energy consumption is reduced by applying the *nmin adaptation* method in ensembles of Hoeffding trees; ii) and second, to see how much accuracy is affected in case of this energy reduction. To fulfill those purposes, we have run a set of five different ensemble algorithms of Hoeffding Trees, with and without *nmin adaptation*, and compared their energy consumption and accuracy. We have tested these algorithms over six synthetic datasets and five real world datasets, described in Table 10.1.

The ensembles used for this paper are: LeveragingBag, OCBoost, OnlineAccuracyUpdatedEnsemble, OzaBag, and OzaBoost. They have already been described in Section 10.4. All of them use the Hoeffding Tree as the base classifier, with and without the *nmin adaptation* method.

The next subsections explain the datasets in more detail, together with the framework and tools for running the algorithms and measuring the energy consumption.

10.5.1 Datasets

The synthetic datasets have been generated using the already available generators in MOA (Massive Online Analysis) [31]. The real world datasets

Table 10.1: *Synthetic and real world datasets. A_i and A_f represent the number of nominal and numerical attributes, respectively. The details of each dataset is presented in Section 10.5.1*

Dataset	Instances	A_i	A_f	Class
Synthetic				
RandomTree	1,000,000	5	5	2
Waveform	1,000,000	0	21	3
RandomRBF	1,000,000	0	10	2
LED	1,000,000	24	0	10
Hyperplane	1,000,000	0	10	2
Agrawal	1,000,000	3	6	2
Real World				
airline	539,383	4	3	2
electricity	45,312	1	6	2
poker	829,201	5	5	10
CICIDS	461,802	78	5	6
forest	581,012	40	10	7

are publicly available online. Each source is detailed with the explanation of the dataset.

RandomTree The random tree dataset is inspired in the dataset proposed by the original authors of the Hoeffding Tree algorithm [1]. The idea is to build a decision tree, by splitting on random attributes, and assigning random values to the leaves. The new examples are then sorted through the tree and labeled based on the values at the leaves.

waveform This dataset is from the UCI repository [32]. A wave is generated as a combination of two or three waves. The task if to predict which one of the three waves the instance represents.

RBF The radial based function (RBF) dataset is created by selecting a number of centroids, each with a random center, class label and weight. Each new example randomly selects a center, considering that centers with higher weights are more likely to be chosen. The chosen centroid represents the class of the example. More details are given by [33].

LED The attributes of this dataset represent each segment of a digit on a LED display. The goal is to predict which is the digit based on the segments, where there is a 10% chance for each attribute to be inverted [34].

Hyperplane The hyperplane dataset is generated by creating a set of points that satisfies $\sum_{i=1}^{d} w_i x_i = w_0$,where x_i is the coordinate for each point [16].

Agrawal The function generates one of ten different predefined loan functions. The dataset is described in the original paper [35].

Airline The airline dataset [36] predicts if a given flight will be delayed based on attributes such as airport of origin and airline.

Electricity The electricity dataset is originally described in [37], and is frequently used in the study of performance comparisons. Each instance represents the change of the electricity price based on different attributes such as day of the week, based on the Australian New South Wales Electricity Market.

Poker The poker dataset is a normalized dataset available from the UCI repository. Each instance represents a hand consisting of five playing cards, where each card has two attributes; suit and rank.

Forest The forest dataset contains the actual forest cover type for a given observation of 30 x 30 meter cells[2].

CICIDS The CICIDS dataset is a cybersecurity dataset from 2017 where the task is to detect intrusions [38].

10.5.2 Environment

The framework used to run the algorithms is MOA (Massive Online Analysis). MOA has the most updated data stream mining algorithms, and is widely used in the field. To evaluate the algorithms in terms of accuracy, we have used the *Evaluate Prequential* option, which tests and then trains on subsets of data. This allows to see the accuracy increase in real time.

Measuring energy consumption is a complicated task. There are no simple ways to measure energy, since there are many hardware variables involved. Based on previous work and experience, we believe that the most

[2] `https://archive.ics.uci.edu/ml/datasets/Covertype`

straight forward approach to measure energy today is by using an interface provided by Intel called RAPL [39]. This interface allows the user to access specific hardware counters available in the processor, which saves energy related measurements of the processor and the DRAM. This approach does not introduce any overhead (in comparison to other approaches such as simulation based), and allows for real time energy measurements. This matches with the requirements in data stream mining. To use the Intel RAPL interface, we use the tool already available: Intel Power Gadget[3]. This tool allows for an energy and power monitoring during the execution of a particular program or script. Since the energy or power is not isolated for that particular program, we have ensured that only our MOA experiments where running on the machine at the time of the measurements. However, since we are aware that there are always programs running in the background, we focus on portraying relative values between different algorithms/methods. For this particular paper, our objective is to understand the difference in energy between using the nmin adaptation method and not using the method. Thus, we focus on the difference between those setups, rather than on absolute values of energy.

All the experiments have been run on a machine with an 3.5 GHz Intel Core i7, with 16GB of RAM, running OSX. We ran the experiments five times and averaged the results.

10.5.3 Reproducibility

The *nmin adaptation* method has been implemented in MOA, to make it openly available once the paper has been reviewed. We have used the implementation in MOA of the ensembles of Hoeffding trees, choosing the Hoeffding tree with nmin adaptation as the base classifier. In order to increase the reproducibility of our results, we have made available the code of the *nmin adaptation*, together with the code of all the experiments, the code used to create the tables and plots, and the datasets. It can be obtained in the following link: `https://www.dropbox.com/sh/ahfsp4i99vd1dmc/AABtjEc4EDkfogS1ZMbz1ofea?dl=0`. At the moment is a private repository, but it will be publicly available in GitHub once the paper has been reviewed.

[3] `https://software.intel.com/en-us/articles/intel-power-gadget-20`

10.6 Results and Discussion

The accuracy and energy consumption results of running algorithms LeveragingBag, OCBoost, OnlineAccuracyUpdatedEnsemble, OZaBag, and OzaBag, on the datasets from Table 10.1, are presented in Tables 10.2 and 10.3, respectively.

Table 10.2: *Accuracy results of running LeveragingBag, OCBoost, OnlineAccuracyUpdatedEnsemble (OAUE), OzaBag, and OzaBoost on ensembles of Hoeffding trees, with and without nmin. Accuracy is measured as the percentage of correctly classified instances. The Nmin and NoNmin columns represent running the algorithm with and without nmin adaptation respectively. Highest accuracy per dataset and algorithm are presented in bold*

	LeveragingBag		OCBoost		OAUE		OzaBag		OzaBoost	
Dataset	nmin	Baseline	nmin	Baseline	nmin	Baseline	nmin	Baseline	nmin	Baseline
Agrawal	**94.84**	94.79	93.47	**93.77**	94.97	**95.01**	94.93	**95.09**	**94.16**	93.13
CICIDS	99.63	**99.75**	48.48	**48.53**	99.38	**99.43**	99.38	**99.51**	**99.43**	99.38
Hyperplane	**90.69**	90.46	**90.43**	89.90	**91.96**	91.93	**90.83**	90.68	**90.40**	90.01
LED	**74.02**	73.92	**17.41**	17.32	73.92	**73.96**	73.89	**73.93**	**73.98**	73.81
RandomRBF	95.08	**95.30**	91.99	**92.26**	92.17	**92.96**	93.43	**93.99**	93.85	**93.93**
RandomTree	97.28	**97.85**	93.11	**93.28**	94.39	**94.72**	94.47	**95.12**	95.81	**96.77**
Waveform	85.77	**85.85**	55.66	**55.85**	80.39	**80.43**	**85.52**	**85.52**	84.52	**84.53**
airline	63.33	**63.49**	**66.09**	66.03	68.55	**68.69**	64.92	**64.93**	**63.46**	63.32
Forest	88.90	**92.50**	73.59	**75.44**	84.34	**85.31**	80.89	**83.93**	87.01	**90.71**
Elec	89.84	**90.44**	89.06	**91.24**	81.54	**83.34**	81.66	**84.08**	86.78	**87.76**
poker	77.54	**88.19**	75.61	**79.87**	69.82	**71.55**	73.92	**83.68**	85.39	**88.40**

Table 10.4 presents the difference in accuracy and energy consumption between running the algorithm with and without *nmin adaptation*, our proposed method. The column $\Delta\%$ details the difference, in percentage, between the accuracy of running the algorithm with and without *nmin adaptation*. A negative value represents that the algorithm with nmin adaptation obtained lower accuracy than the algorithm with the original implementation. The column $\Delta(J)$ presents the difference, in percentage, between the energy consumption of running the algorithm with and without *nmin adaptation*. The lower the value the better, since it means that we reduced the energy consumption by that amount.

Looking at Table 10.4, we can see how for all algorithms and for all datasets *nmin adaptation* reduced the energy consumption by 21 percent, up to a 41 percent. Accuracy is affected by less than 1 percent on average.

Table 10.3: *Energy consumption results of running LeveragingBag, OCBoost, OnlineAccuracyUpdatedEnsemble (OAUE), OzaBag, and OzaBoost on ensembles of Hoeffding trees, with and without nmin. The energy is measured in joules(J). Lowest energy consumption per dataset and algorithm are presented in bold. The Nmin and NoNmin columns represent running the algorithm with and without nmin adaptation respectively*

	LeveragingBag		OCBoost		OAUE		OzaBag		OzaBoost	
Dataset	nmin	Baseline	nmin	Baseline	nmin	Baseline	nmin	Baseline	nmin	Baseline
Agrawal	**5121.81**	6856.58	**1203.68**	1522.84	**1326.48**	1374.64	**1236.72**	1352.17	**2175.24**	3018.05
CICIDS	**1448.66**	1709.49	**1511.78**	1658.37	**1415.23**	1528.05	**1193.95**	1362.44	**1463.70**	1621.17
Hyperplane	**8391.08**	14370.83	**1009.45**	1364.35	**856.30**	1246.84	**1233.10**	1815.75	**1307.83**	1881.81
LED	**1854.47**	2325.33	**1614.94**	1865.63	**1645.07**	1772.10	**1218.56**	1397.10	**1280.74**	1507.98
RandomRBF	**7389.42**	10617.94	**1070.32**	1347.56	**1362.92**	1796.12	**1614.04**	2111.28	**1637.68**	2293.95
RandomTree	**7900.56**	12534.89	**1510.23**	1791.32	**1873.98**	2452.21	**1887.13**	2555.93	**2465.27**	3942.07
Waveform	**9164.32**	15182.30	**1759.72**	2298.32	**1371.15**	1533.26	**1544.43**	2190.14	**1907.51**	2593.89
airline	**10845.63**	12028.00	**5015.23**	6310.04	**1789.47**	3004.81	**6127.37**	7132.72	**7057.96**	7907.50
Forest	**2061.23**	2801.53	**1619.14**	1919.68	**1820.17**	1969.78	**1347.59**	2156.80	**1693.18**	2496.60
Elec	**148.72**	183.86	**93.18**	111.81	**104.77**	121.84	**102.15**	115.40	**95.94**	118.15
poker	**744.47**	1220.16	**917.18**	1023.73	**759.18**	844.75	**641.70**	934.99	**801.12**	1051.66

The poker dataset obtained the highest difference in accuracy between both methods, up to a 10 percent. However, the other datasets show how the original ensembles of Hoeffding trees and the version with *nmin adaptation* are comparable in terms of accuracy.

Figure 10.10 shows how accuracy increases with the number of instances. Each subplot represents all algorithms averaged for that particular dataset. Our goal with this figure is to show the difference between running the ensembles with and without *nmin adaptation* in terms of accuracy. We can observe how for most of the datasets (Agrawal, Hyperplane, LED, RandomRBF, RandomTree, Waveform, and airline) the difference is indiscernible, suggesting that the algorithm performs equally well for both the nmin adaptation approach and the standard approach. For the poker and forest dataset, we can see how *nmin adaptation* obtains lower accuracy than the standard approach. This was also visible in Table 10.2.

Figure 10.11 shows the energy consumption per dataset, per algorithm, with and without *nmin adaptation*. This figure clearly shows how all algorithms with *nmin adaptation* consume less energy than the standard version. This occurs in all datasets, even in datasets where the accuracy is higher for the *nmin adaptation* approach (e.g. Agrawal dataset, OzaBoost algorithm).

Table 10.4: *Difference in accuracy(Δ%) and energy consumption Δ(J) between the algorithms running with and without nmin adaptation. The difference is measured as the percentage between both approaches. For the Δ(%) column, the higher the value the better, since it means that the nmin adaptation approach obtained higher accuracy than the non-nmin adaptation. For the Δ(J) the lower the better, meaning that we reduced the energy consumption by that percent.*

	LeveragingBag		OCBoost		OAUE		OzaBag		OzaBoost	
Dataset	Δ%	Δ(J)	Δ%	Δ(J)	Δ%	Δ(J)	Δ%	Δ(J)	Δ%	Δ(J)
Agrawal	0.05	-25.30	-0.30	-20.96	-0.04	-3.50	-0.16	-8.54	1.04	-27.93
CICIDS	-0.12	-15.26	-0.05	-8.84	-0.05	-7.38	-0.13	-12.37	0.06	-9.71
Hyperplane	0.23	-41.61	0.53	-26.01	0.03	-31.32	0.15	-32.09	0.39	-30.50
LED	0.11	-20.25	0.09	-13.44	-0.05	-7.17	-0.04	-12.78	0.17	-15.07
RandomRBF	-0.22	-30.41	-0.27	-20.57	-0.79	-24.12	-0.56	-23.55	-0.08	-28.61
RandomTree	-0.56	-36.97	-0.17	-15.69	-0.33	-23.58	-0.65	-26.17	-0.96	-37.46
Waveform	-0.08	-39.64	-0.19	-23.43	-0.03	-10.57	0.00	-29.48	-0.00	-26.46
airline	-0.16	-9.83	0.06	-20.52	-0.14	-40.45	-0.01	-14.09	0.14	-10.74
Forest	-3.60	-26.42	-1.85	-15.66	-0.97	-7.60	-3.04	-37.52	-3.70	-32.18
Elec	-0.60	-19.11	-2.18	-16.67	-1.80	-14.01	-2.42	-11.48	-0.98	-18.80
poker	-10.65	-38.99	-4.26	-10.41	-1.73	-10.13	-9.76	-31.37	-3.01	-23.82
Average	-1.42	-27.62	-0.78	-17.47	-0.54	-16.35	-1.51	-21.77	-0.63	-23.75

At a first glance, and without analyzing the results in depth, one could think that a higher accuracy requires a higher energy cost. However, our results show that there is not a direct relationship between accuracy increase and an increase in energy consumption. Looking at Table 10.4, for instance at the OzaBoost algorithm, we can see how there is a high energy decrease (30 percent for the Hyperplane dataset) while still obtaining a higher accuracy (0.39 percent). Another algorithm with a high energy decrease is the OAUE for the airline dataset, where we decrease 40% the energy and accuracy is affected by 0.14 percent. A similar energy reduction is obtained in the poker dataset, for the LeveragingBag algorithm. However in this case the accuracy is significantly affected by 10%. These results are very promising, since they suggest that there is not a correlation between energy consumption and accuracy. Thus, there are ways, such as our current approach, to reduce the energy consumption of an algorithm without having to sacrifice accuracy. The importance lies on finding those hotspots that make the algorithm consume energy inefficiently.

In overall, our results show how *nmin adaptation* is able to reduce the

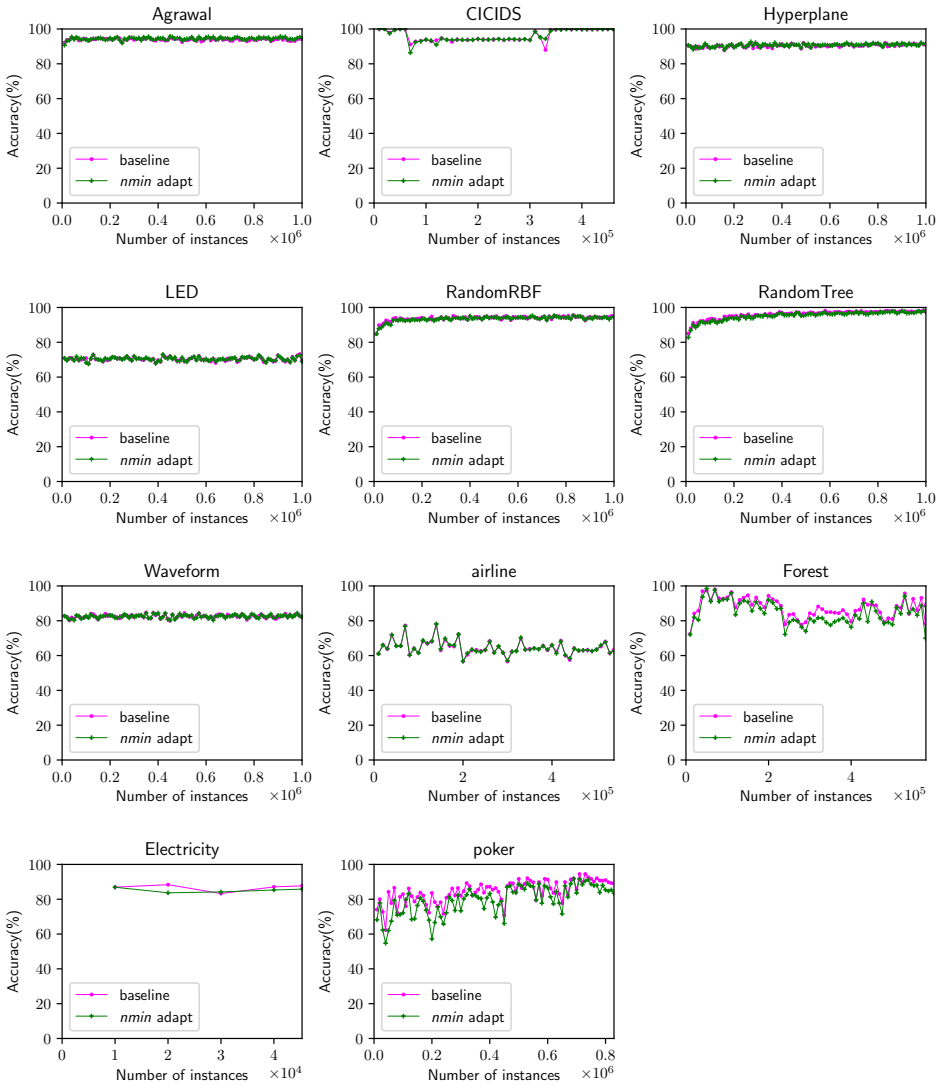

Figure 10.10: *Accuracy comparison between running the algorithms with and without nmin adaptation on all datasets. The results of each dataset are averaged for all algorithms*

energy consumption by 21 percent on average, affecting accuracy by less than one percent on average. This demonstrates that our solution is able to create more energy efficient ensembles of Hoeffding trees trading off less than one percent of accuracy.

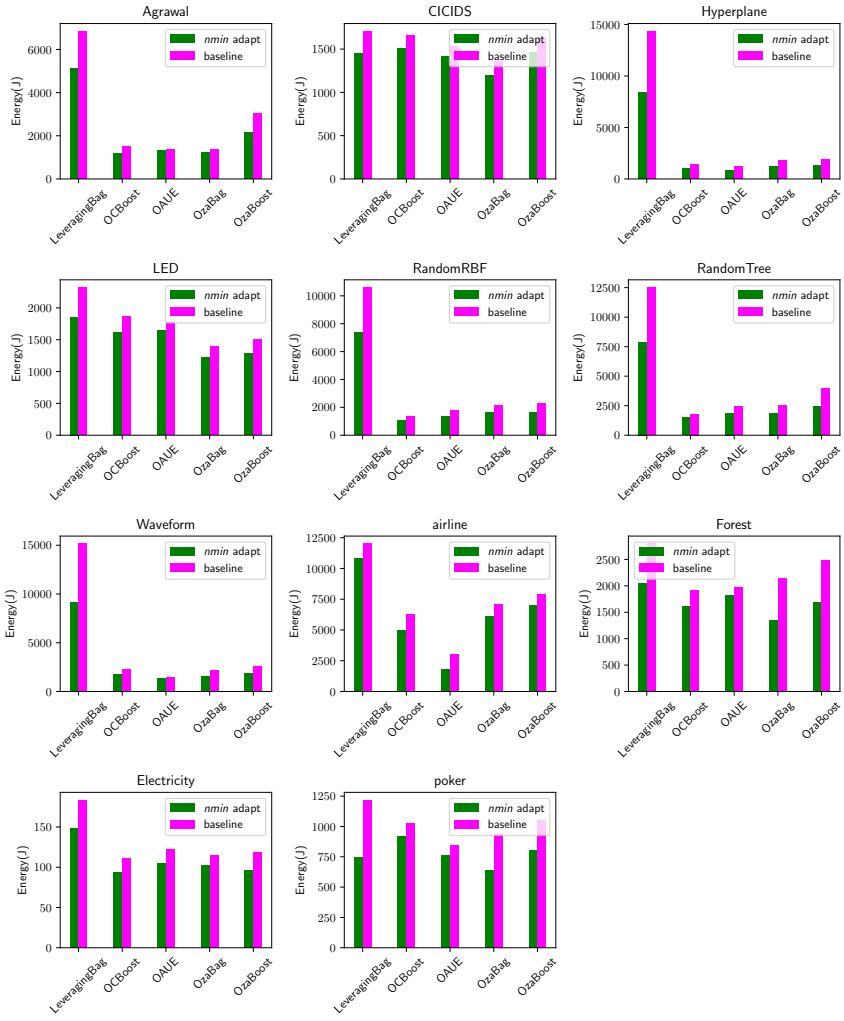

Figure 10.11: *Energy consumption for all datasets and all algorithms, with and without nmin adaptation*

10.7 Conclusions

This paper presents a simple, yet effective approach to reduce the energy consumption of ensembles of Hoeffding tree algorithms. This method, *nmin adaptation*, estimates the batch size of instances needed to check for a split, individually for each node. Thus, allowing the Hoeffding tree to grow faster in those branches where the confidence for choosing the best attribute is

higher, and delaying the split in those branches where the confidence is lower.

We also present a generic approach to build theoretical energy models for any class of algorithms, together with energy models for the ensembles of Hoeffding trees used in this paper.

We conduct a set of experiments where we compare the accuracy and energy consumption of five ensembles of Hoeffding trees with and without nmin, on 11 publicly available datasets. The results show that *nmin adaptation* reduces the energy consumption significantly, on all datasets (up to a 41%, with 21% on average). This is achieved by trading off a few percentages of accuracy (up to a 10% for one particular dataset, on average of less than 1%). Our results also show that there is no clear correlation between a higher accuracy and a higher energy consumption. Thus, opening the possibility for more research to create more energy efficient algorithms without sacrificing accuracy.

While data stream mining approaches are focused on running on edge devices, still there has not been much research that tackle the energy efficiency of the approaches, which is a key variable for these devices. This study provides one step forward into achieving more energy efficient algorithms that are able to run in embedded devices. For future work, we plan on designing more energy efficient Hoeffding trees that are able to compete with high accurate current approaches[40].

10.8 References

[1] P. Domingos and G. Hulten. "Mining high-speed data streams". In: *Proc. 6th SIGKDD Int'l Conf. on Knowledge discovery and data mining*. 2000, pp. 71–80.

[2] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, and V. Boeva. "Hoeffding Trees with nmin adaptation". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2018, pp. 70–79.

[3] A. Bifet, G. Holmes, and B. Pfahringer. "Leveraging bagging for evolving data streams". In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2010, pp. 135–150.

[4] R. Pelossof, M. Jones, I. Vovsha, and C. Rudin. “Online coordinate boosting”. In: *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE. 2009, pp. 1354–1361.

[5] D. Brzeziński and J. Stefanowski. “Accuracy updated ensemble for data streams with concept drift”. In: *International conference on hybrid artificial intelligence systems*. Springer. 2011, pp. 155–163.

[6] N. C. Oza and S. Russell. “Online Bagging and Boosting”. In: *In Artificial Intelligence and Statistics 2001*. Morgan Kaufmann, 2001, pp. 105–112.

[7] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[8] M. Horowitz. “1.1 computing’s energy problem (and what we can do about it)”. In: *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*. IEEE. 2014, pp. 10–14.

[9] J. G. Koomey, S. Berard, M. Sanchez, and H. Wong. “Assessing trends in the electrical efficiency of computation over time”. In: *IEEE Annals of the History of Computing* 17 (2009).

[10] S. Han, J. Pool, J. Tran, and W. Dally. “Learning both weights and connections for efficient neural network”. In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[11] S. Han, H. Mao, and W. J. Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding”. In: *arXiv preprint arXiv:1510.00149* (2015).

[12] C. F. Rodrigues, G. Riley, and M. Luján. “Fine-grained energy profiling for deep convolutional neural networks on the Jetson TX1”. In: *Workload Characterization (IISWC), 2017 IEEE International Symposium on*. IEEE. 2017, pp. 114–115.

[13] K. Gauen, R. Rangan, A. Mohan, Y.-H. Lu, W. Liu, and A. C. Berg. “Low-power image recognition challenge”. In: *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE. 2017, pp. 99–104.

[14] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks”. In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138.

[15] C. F. Rodrigues, G. Riley, and M. Luján. "SyNERGY: An energy measurement and prediction framework for Convolutional Neural Networks on Jetson TX1". In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2018, pp. 375–382.

[16] G. Hulten, L. Spencer, and P. Domingos. "Mining time-changing data streams". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 97–106.

[17] A. Bifet and R. Gavalda. "Learning from time-changing data with adaptive windowing". In: *Proceedings of the 2007 SIAM international conference on data mining*. SIAM. 2007, pp. 443–448.

[18] A. Bifet and R. Gavaldà. "Adaptive learning from evolving data streams". In: *International Symposium on Intelligent Data Analysis*. Springer. 2009, pp. 249–260.

[19] N. Kourtellis, G. D. F. Morales, A. Bifet, and A. Murdopo. "VHT: Vertical Hoeffding Tree". In: *arXiv preprint arXiv:1607.08325* (2016).

[20] Y. Ben-Haim and E. Tom-Tov. "A streaming parallel decision tree algorithm". In: *Journal of Machine Learning Research* 11.Feb (2010), pp. 849–872.

[21] D. Marrón, E. Ayguadé, J. R. Herrero, J. Read, and A. Bifet. "Low-latency multi-threaded ensemble learning for dynamic big data streams". In: *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE. 2017, pp. 223–232.

[22] V. Losing, B. Hammer, and H. Wersing. "KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift". In: *IEEE 16th International Conference on Data Mining (ICDM)*. Dec. 2016, pp. 291–300.

[23] V. Losing, H. Wersing, and B. Hammer. "Enhancing Very Fast Decision Trees with Local Split-Time Predictions". In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2018, pp. 287–296.

[24] W. Hoeffding. “Probability inequalities for sums of bounded random variables”. In: *Journal of the American statistical association* 58.301 (1963), pp. 13–30.

[25] L. Breiman. “Bagging Predictors”. In: *Machine Learning* 24.2 (1996), pp. 123–140.

[26] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. “New ensemble methods for evolving data streams”. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*. 2009, pp. 139–148.

[27] N. C. Oza. “Online bagging and boosting”. In: *2005 IEEE international conference on systems, man and cybernetics*. Vol. 3. Ieee. 2005, pp. 2340–2345.

[28] N. C. Oza and S. Russell. “Experimental comparisons of online and batch versions of bagging and boosting”. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 359–364.

[29] Y. Freund and R. E. Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: *Computational Learning Theory, Second European Conference, EuroCOLT ’95, Barcelona, Spain, March 13-15, 1995, Proceedings*. 1995, pp. 23–37.

[30] D. Brzezinski and J. Stefanowski. “Combining block-based and online methods in learning ensembles from concept drifting data streams”. In: *Information Sciences* 265 (2014), pp. 50–67.

[31] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. “MOA: Massive online analysis”. In: *The Journal of Machine Learning Research* 11 (2010), pp. 1601–1604.

[32] D. Dheeru and E. Karra Taniskidou. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu/ml`.

[33] A. Bifet, J. Zhang, W. Fan, C. He, J. Zhang, J. Qian, G. Holmes, and B. Pfahringer. “Extremely Fast Decision Tree Mining for Evolving Data Streams”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2017, pp. 1733–1742.

[34] L. Breiman. *Classification and regression trees*. Routledge, 2017.

[35] R. Agrawal, T. Imielinski, and A. Swami. "Database mining: A performance perspective". In: *IEEE transactions on knowledge and data engineering* 5.6 (1993), pp. 914–925.

[36] E. Ikonomovska. *Datasets*. Retrieved from `http://kt.ijs.si/elena_ikonomovska/data.html`. Online; accessed 1 December 2019. 2013.

[37] M. Harries. *Splice-2 comparative evaluation: Electricity pricing*. Tech. rep. 1999.

[38] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization." In: *ICISSP*. 2018, pp. 108–116.

[39] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. "RAPL: Memory power estimation and capping". In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. Aug. 2010, pp. 189–194. DOI: `10.1145/1840845.1840883`.

[40] C. Manapragada, G. I. Webb, and M. Salehi. "Extremely fast decision tree". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2018, pp. 1953–1962.

11

Green Accelerated Hoeffding Tree

Eva García-Martín, Albert Bifet, Niklas Lavesson

Abstract

For the past years, the main concern in machine learning had been to create highly accurate models, without considering the high computational requirements involved. Stream mining algorithms are able to produce highly accurate models in real time, without strong computational demands. This is the case of the Hoeffding tree algorithm. Recent extensions to this algorithm, such as the Extremely Very Fast Decision Tree (EFDT), focus on increasing predictive accuracy, but at the cost of a higher energy consumption. This paper presents the Green Accelerated Hoeffding Tree (GAHT) algorithm, which is able to achieve same levels of accuracy as the latest EFDT, while reducing its energy consumption by 27 percent with minimal effect on accuracy.

11.1 Introduction

Data stream mining algorithms perform real time predictions with low computational requirements, with the end goal to run on edge devices. This is the case of the Hoeffding Tree (HT) algorithm [1], a pioneer algorithm for building decision trees for data streams with theoretical guarantees. However, current research suggests that still the main focus of these class of algorithms is to obtain a higher predictive accuracy than their predecessors [2].

Recent extensions to this algorithm, such as ensembles of Hoeffding trees [3] and the Extremely Fast Decision Tree (EFDT) [4] build highly accurate Hoeffding trees but at a high energy cost.

This paper addresses that challenge by presenting the Green Accelerated Hoeffding Tree (GAHT), an extension of the EFDT algorithm that is able to achieve the same accuracy results while reducing its energy consumption by 27 percent on average. The GAHT grows the tree individually for each

branch, setting an individual splitting criteria for each node based on the distribution of the number of instances over the leaves of the tree. This allows for having different nodes with different energy efficiency levels, that combined and averaged result in an energy reduction of the overall tree.

The GAHT is also able to build smaller trees, which is desirable for running these algorithms in memory constrained devices. The results of this paper show that it is possible to build energy efficient stream mining algorithms with competitive predictive performance guarantees.

11.2 Background

This section explains in detail the VFDT and EFDT algorithms, while the GAHT is explained in Section 11.4.

11.2.1 Very Fast Decision Tree

The Very Fast Decision Tree is the implementation of the Hoeffding Tree algorithm, proposed by [1]. The VFDT was the first algorithm that was able to mine from infinite streams of data, requiring low computational power.

The algorithm is able to create a decision tree in real time obtaining similar performance as offline decision trees, with theoretical guarantees. This is achieved by saving the necessary information (statistics) from the different instances observed at each node. Once $nmin$ instances are observed at a leaf, the algorithm calculates the information gain (entropy) for each attribute, using the aforementioned statistics. If $\Delta\overline{G} > \epsilon$, i.e. the difference between the information gain from the best and the second best attribute ($\Delta\overline{G}$) is higher than the Hoeffidng Bound [5](ϵ), then a split occurs, and the leaf substituted by the node with the best attribute. The Hoeffding bound(ϵ) is defined as:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{11.1}$$

and it states that the split on the best attribute having observed n number of examples, will be the same as if the algorithm had observed an infinite number of examples, with probability $1 - \delta$. On the other hand, if $\Delta\overline{G} < \epsilon < \tau$, a tie occurs. This happens when the two top attributes are equally good, thus the tree splits on any of those.

11.2.2 Extremely Fast Decision Tree

The EFDT is the implementation of the Hoeffding Anytime Tree, which was recently published at KDD [4]. The EFDT is able to create a Hoeffding tree with higher predictive performance than the VFDT. This is achieved by having a less restrictive split criterion, allowing the tree to grow faster; and by reevaluating the already split nodes to try to resemble more the asymptotic batch decision tree. While the EFDT is able to converge faster to a higher accuracy, it does this at a higher cost of energy consumption, compared to the original VFDT.

11.3 Related Work

Energy efficiency has been studied in computer architecture for several decades, with a current focus on designing processors that can perform more operations under the same power budget [6]. In the field of machine learning, there is a current trend that focuses on reducing the energy consumption of both training and inference, to be able to run models on the edge [7]. This is the case for deep learning [8–11], an area where they have also created *The Low Power Image Recognition Challenge* (LPIRC) [12], a competition where they present the best technologies that can detect objects with high precision and low energy consumption.

Data stream mining algorithms started to gain importance in the year 2000, with the publication of the VFDT [1] algorithm. This algorithm was the first that could analyze an infinite stream of data in constant time per example. The VFDT was then extended to handle concept drift [13, 14], that is change in the data source. Ensembles of Hoeffding trees were introduced to be able to achieve a higher predictive accuracy. Models such as Leveraging Bagging [3], Online Coordinate Boosting [15], and the novel Adaptive Random Forest [16], that can also handle concept drift.

The latest VFDT algorithm is the already mentioned Extremely Fast Decision Tree (EFDT) (Section 11.2), which obtains higher accuracy compared to the original VFDT, but taking longer to run [4]. In relation to energy efficiency, the Vertical Hoeffding Tree (VHT) [17] algorithm was introduced as a parallel version of the VFDT. [18] proposed a parallel version of random forests of Hoeffding trees with specific hardware configurations. A recently published study has modified the splitting criteria of the VFDT

by adapting the value of the batch size of instances to reduce its energy consumption [19]. Another similar approach [20] also adapts the value of the batch size, obtaining higher accuracy and higher energy consumption than [19]. In this paper we don't compare the GAHT against the mentioned solutions because they are less accurate than the EFDT and the goal is to achieve a competitive state-of-the-art accuracy.

The GAHT, our proposed algorithm, is a more energy efficient extension of the EFDT. This is achieved by applying the same splitting criteria as the EFDT only on a set of selected nodes, and deactivating another set of less visited nodes. We explain it in detail in the following section.

11.4 Green Accelerated Hoeffding Tree

This section presents the Green Accelerated Hoeffding Tree algorithm, the main contribution of this paper. The GAHT, presented in Alg. 16, follows a per-node splitting criteria that varies depending on the observed data. The goal of the GAHT is to present an approach that can obtain competitive accuracy results, while still staying within a constrained energy budget. In particular, our goal is to achieve the same accuracy as the EFDT, which is the most novel Hoeffding tree, but at a lower energy cost.

For that, we follow a similar reasoning as the authors of the *nmin adaptation* method [19], that is to have a unique parameter value for each node in the tree, since different nodes have observed different samples of the incoming data. Looking at Alg. 16, the GAHT reads an instance, traverses the tree, and for each instance it calculates the fraction of instances seen at that leaf (considering the amount of instances observed since that leaf was created, and the number of leaves); as follows:

$$fraction = \frac{n_l}{n_{since\_creation}/n_{leaves}} \tag{11.2}$$

If $fraction$ is lower than the $deactivateThreshold$, that leaf is deactivated. When a leaf is deactivated the statistics are still stored, but the leaf is not allowed to grow. On the other hand, if $fraction$ is higher than the $growFastThreshold$, then that leaf is set to the EFDT splitting criteria, making faster splits on that branch. That is, instead of comparing the information gain of the two best attributes, the algorithm compares the information gain of the best attribute against the null split. The null split stands for the information gain without creating a split.

We have implemented the GAHT so that the user can choose the values of parameters *deactivateThreshold* and *growFastThreshold*. For these experiments we set `deactivateThreshold = 0.01` and `growFastThreshold = 2`, to deactivate leaves observed less than 1 percent of the time, and grow faster the leaves observed twice the average. In the case where there are restrictions on energy consumption, the user can increase the value of *deactivateThreshold*, since deactivating more nodes will reduce the energy consumption.

To achieve reproducibility of our results, the code of the GAHT (as a part of MOA), together with the code for the experiments, tables and plots, is available in the following link: `https://www.dropbox.com/sh/a5rozyjvbizsctu/AAC1pISlO3aD7F9HoRyKCK4ba?dl=0`. It will he publicly available in GitHub once the paper has been reviewed.

11.5 Experimental Design

We designed a set of experiments to evaluate the accuracy and energy consumption of the GAHT, compared to the EFDT and the VFDT. We also compared our solution to the VFDT to understand how much extra energy is being consumed to achieve the higher energy consumption. We have decided not to compare against the two other novel Hoeffding tree extensions (mentioned in Section 11.3), because the algorithms either output less accuracy than the VFDT [19], or less accuracy than the EFDT [20]. We have also decided not to compare against ensembles of Hoeffding trees due to their high energy consumption (significantly higher than the EFDT). The goal of the GAHT is to extend the EFDT to be more energy efficient, thus presenting, to the best of our knowledge, the Hoeffding tree extension with a single tree that outputs the highest accuracy and less energy consumption.

There are a total of 11 datasets used for the experiments, publicly available online, which are described in Table 11.1. The synthetic datasets are generated with MOA, with the default settings. The electricity, poker, airline, forest, and synthetic datasets are better explained in [21], and in the official MOA site: `https://moa.cms.waikato.ac.nz/datasets/`. The CICIDS is an intrusion detection dataset from 2017 [22].

We have used the VFDT algorithm version available in MOA. For the EFDT, we have used the implementation in MOA provided by the original

Algorithm 16 Green Accelerated Hoeffding Tree. **Symbols:**HT: Initial tree; X: set of attributes; $G(\cdot)$: split evaluation function; τ: tie threshold; $nmin$: batch size

while stream is not empty **do**
  Read instance
  Traverse the tree using HT
  Update statistics at leaf l {Attributes}
  Increment n_l: instances seen at l
  $fraction = \frac{n_l}{n_{since\_creation}/n_{leaves}}$
  if $fraction <$ deactivateThreshold **then**
    DeactivateLeaf
  else if $fraction >$ growFastThreshold **then**
    growFast $\leftarrow$ True {EFDT criteria on l}
  end if
  if $nmin \leq n_l$ **then**
    Compute $\overline{G_l}(X_i)$ for each attribute X_i
    if growFast == True **then**
      X_b = null split
    else
      X_b = second best attribute
    end if
    $\Delta\overline{G} = \overline{G}_l(X_a) - \overline{G}_l(X_b)$
    if $(\Delta\overline{G} > \epsilon)$ or $(\epsilon < \tau)$ **then**
      Split $\leftarrow$ True
    end if
    if Split==True **then**
      CreateChildren(l)
    else
      Disable att $\{X_p|(\overline{G}_l(X_p) - \overline{G}_l(X_a)) > \epsilon\}$
    end if
  end if
end while

authors[1]. We set `deactivateThreshold=0.01` and `growFastThreshold=2` for the GAHT, after having tested different parameter setups. Thus, we deactivate all leaves that have been observed less than 1 percent of the total observations; and we split faster on the leaves that observed at least twice

[1] `https://github.com/chaitanya-m/kdd2018`

Table 11.1: *Synthetic and real world datasets used in the experiments. A_i and A_f represent the number of nominal and numerical attributes, respectively. Abbrv. presents the abbreviation used for Tables 11.2 and 11.3.*

Dataset	Instances	Abbrv.	A_i	A_f	Class
Synthetic					
RandomTree	1,000,000	RTRee	5	5	2
Waveform	1,000,000	Wave	0	21	3
RandomRBF	1,000,000	RBF	0	10	2
LED	1,000,000	LED	24	0	10
Hyperplane	1,000,000	Hyper	0	10	2
Agrawal	1,000,000	Agrawal	3	6	2
Real World					
Airline	539,383	Airline	4	3	2
Electricity	45,312	Elec	1	6	2
Poker	829,201	Poker	5	5	10
CICIDS	461,802	CICIDS	78	5	6
Forest	581,012	Forest	40	10	7

* AWS limited to 1M instances.

as many instances as the average leaf. After some experimentations The accuracy was evaluated using the *Evaluate Prequential* function from MOA, where each instance is first used to test the model and then to train it. The frequency of how often the performance is calculated is set to 100k for all datasets, except for the *Electricity* dataset, where we set it to 4k.

In this paper we have measured the energy consumption using Intel's RAPL interface [23]. This interface accesses specific hardware counters available in the processor, which saves energy related measurements of the processor and the DRAM. To use the Intel RAPL interface we use Intel Power Gadget[2], a tool developed by Intel that outputs power and energy real time measurements. Since Intel Power Gadget does not isolate the energy consumption of a specific program, to estimate the energy consumed by the GAHT, EFDT, and VFDT we have ensured that only our MOA experiments where running on the machine at the time of the measurements. However, since we are aware that there are always programs running in the background, we focus on portraying relative values between different algorithms. All the experiments have been run on a machine with an 3.5 GHz

[2] `https://software.intel.com/en-us/articles/intel-power-gadget-20`

Intel Core i7, with 16GB of RAM, running OSX. We ran the experiments five times and averaged the results.

11.6 Results and Discussion

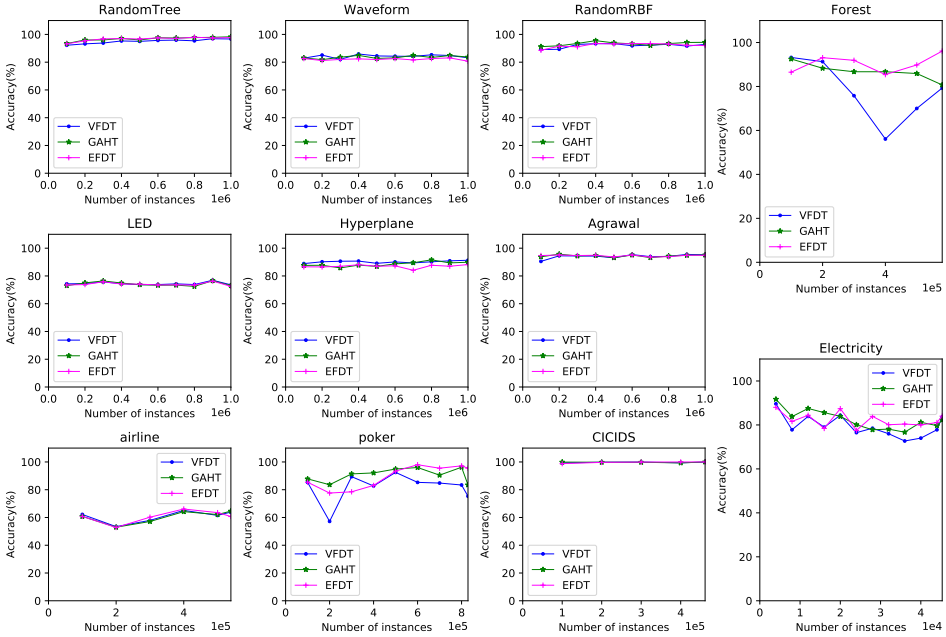

Figure 11.1: *Accuracy comparison between the GAHT, EFDT, VFDT on all datasets. Accuracy was evaluated every 100k instances for all datasets except for the Electricity dataset, where accuracy was evaluated every 40k instances.*

The energy and accuracy results of running the GAHT, EFDT, and VFDT are presented in Table 11.2, and summarized in Table 11.3. The results show that the GAHT achieves similar or higher accuracy (Fig. 11.1) than the EFDT on almost all datasets, while reducing the energy consumption by an average of 27%. GAHT obtains a lower energy consumption than the EFDT on all datasets, with a maximum energy reduction (Table 11.3) of 41.7% on the LED dataset, and a minimum energy reduction of 16.9% on the Electricity dataset. Figures 11.2 and 11.3 give a better understanding about the differences in accuracy and energy consumption between the three algorithms. In particular, Fig. 11.2 shows that the GAHT and the EFDT obtain similar levels of accuracy, and higher levels of accuracy than the

VFDT. The VFDT has a larger interquartile range and a lower quartile, suggesting that the accuracy values of the VFDT are lower and more spread around. Fig. 11.2 shows how GAHT consumes significantly less energy than the EFDT. It also shows that the VFDT is the algorithm with the lowest energy consumption, however sacrificing accuracy to achieve so.

Table 11.2: *Accuracy and energy consumption comparison between GAHT, EFDT, and VFDT. Number of nodes, leaves, active leaves are presented for the GAHT, EFDT, and VFDT. Number of inactive leaves and number of fast nodes are presented for the GAHT. Fast nodes are those nodes in the GAHT that have the EFDT splitting criteria. (Modified for this thesis to fit in the page size*

	Accuracy (%)			Total Energy(J)			#Nodes			#Leaves			#Inactive	#Fast
Dataset	GAHT	EFDT	VFDT	GAHT	EFDT	VFDT	GAHT	EFDT	VFDT	GAHT	EFDT	VFDT	GAHT	GAHT
RTree	96.74	96.68	95.01	146.47	192.79	122.98	1988.0	2572.0	1132.0	1239.0	1561.0	724.0	0.0	802.0
Wave	83.60	82.07	84.22	409.18	551.27	167.01	5195.0	4611.0	323.0	2598.0	1544.0	162.0	4.0	3937.0
RBF	93.31	92.19	91.93	195.84	324.63	129.31	3037.0	3567.0	819.0	1519.0	1134.0	410.0	7.0	1915.0
LED	74.18	74.07	74.50	235.18	403.80	139.83	1669.0	2795.0	107.0	835.0	1169.0	54.0	0.0	1068.0
Hyper	88.49	86.94	90.14	190.96	285.67	123.58	3191.0	3513.0	665.0	1596.0	1199.0	333.0	0.0	2325.0
Agrawal	94.44	94.55	94.10	117.71	162.00	96.20	775.0	1713.0	408.0	516.0	1250.0	238.0	4.0	264.0
Airline	60.30	60.68	60.67	226.50	282.93	170.28	19755.0	30917.0	8603.0	18802.0	30315.0	8536.0	0.0	6833.0
Poker	90.67	89.36	81.77	206.00	250.10	137.69	4700.0	2240.0	903.0	2533.0	309.0	492.0	169.0	3806.0
CICIDS	99.76	99.68	99.66	254.60	345.30	241.16	129.0	161.0	63.0	65.0	14.0	32.0	14.0	83.0
Forest	86.80	90.47	77.62	379.64	546.98	260.48	3147.0	2200.0	699.0	1574.0	549.0	350.0	53.0	2709.0
Elec	82.37	82.23	79.48	51.28	61.71	36.44	341.0	270.0	77.0	196.0	144.0	49.0	0.0	255.0

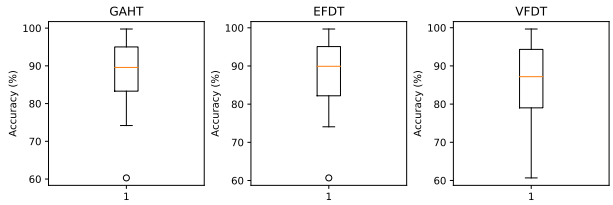

Figure 11.2: *Differences in accuracy for the GAHT, EFDT, and VFDT algorithms*

Fig. 11.4 compares the number of nodes needed by the GAHT and EFDT to achieve a specific accuracy. The GAHT creates smaller trees than the EFDT while obtaining similar or higher accuracy in the majority of datasets, which is essential for running these algorithms in devices with limited memory capacity. The last column of Table 11.2 represents the amount of GAHT nodes set with the EFDT splitting criteria, i.e. fast nodes. The results confirm that having only a set of nodes with the EFDT splitting criteria (GAHT) achieves the same (or higher) accuracy than setting all

Table 11.3: *Difference in accuracy (ΔA) and energy consumption (ΔE) between GAHT, EFDT, and VFDT. The difference is measured as the percentage reduction between the algorithms. A negative value in the energy means that the algorithm reduced the energy by that amount.*

	GAHT vs EFDT		GAHT vs VFDT		EFDT vs VFDT	
Dataset	ΔA (%)	ΔE(%)	ΔA (%)	ΔE(%)	ΔA(%)	ΔE (%)
RTree	0.06	-24.03	1.73	19.10	1.67	56.77
Wave	1.53	-25.77	-0.62	145.00	-2.15	230.08
RBF	1.12	-39.67	1.38	51.45	0.26	151.05
LED	0.11	-41.76	-0.32	68.18	-0.43	188.77
Hyper	1.55	-33.15	-1.65	54.53	-3.20	131.16
Agrawal	-0.11	-27.34	0.34	22.37	0.45	68.41
Airline	-0.38	-19.95	-0.37	33.02	0.01	66.16
Poker	1.31	-17.63	8.90	49.62	7.59	81.65
CICIDS	0.08	-26.27	0.10	5.57	0.02	43.18
Forest	-3.67	-30.59	9.18	45.75	12.85	109.99
Elec	0.14	-16.89	2.89	40.75	2.75	69.35
Average	0.16	-27.55	1.96	48.67	1.80	108.78

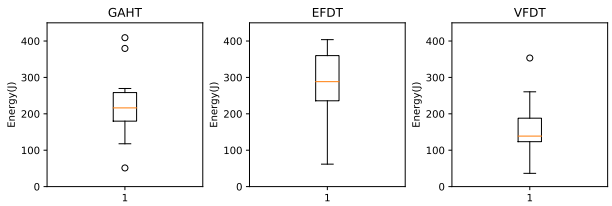

Figure 11.3: *Differences in energy for the GAHT, EFDT, and VFDT algorithms*

nodes with the EFDT criteria (EFDT). For instance, the GAHT creates 40% of fast nodes while the EFDT creates 129% (compared to the GAHT) for the random tree dataset, while outputting the same accuracy.

We finally compare the energy efficiency of the GAHT against the EFDT and the VFDT (Fig. 11.5). We define energy efficiency as the energy proportionality in relation to accuracy, i.e. if the energy spent on building the tree is reflected as an increase in accuracy. The ideal scenario is when an increase in energy consumption creates a proportional increase in accuracy. It is calculated as:

$$\text{energy efficiency} = \frac{energy}{kappa} \tag{11.3}$$

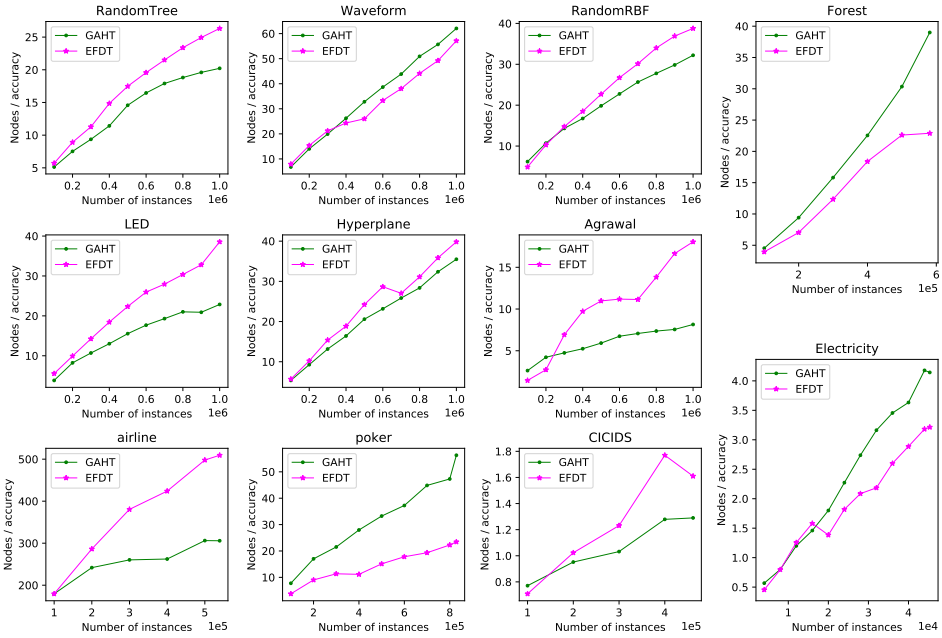

Figure 11.4: *Tree size comparison between the GAHT and the EFDT. The y-axis represents how many nodes are needed to achieve a specific accuracy. The lower the value the better, representing smaller trees with the highest accuracy, useful for embedded devices with memory constraints.*

where *kappa* is the difference in accuracy between the algorithm and if the algorithm would choose the majority class as the label. The lower the value of energy efficiency the better, which shows that less energy is needed to achieve a specific kappa (accuracy).

Fig. 11.5 shows that the GAHT is more energy efficient than the EFDT in all datasets and than the VFDT on 50 percent of the datasets. In scenarios where the energy is the main constraint the VFDT would be the chosen option. However if there is a need for high accuracy but under an energy budget, then the GAHT would be the best possible option.

In overall, our method is able to adapt the splitting criteria at the leaf based on the distribution of the number of instances observed at that particular leaf. This creates smaller, high accuracy, and energy efficient trees.

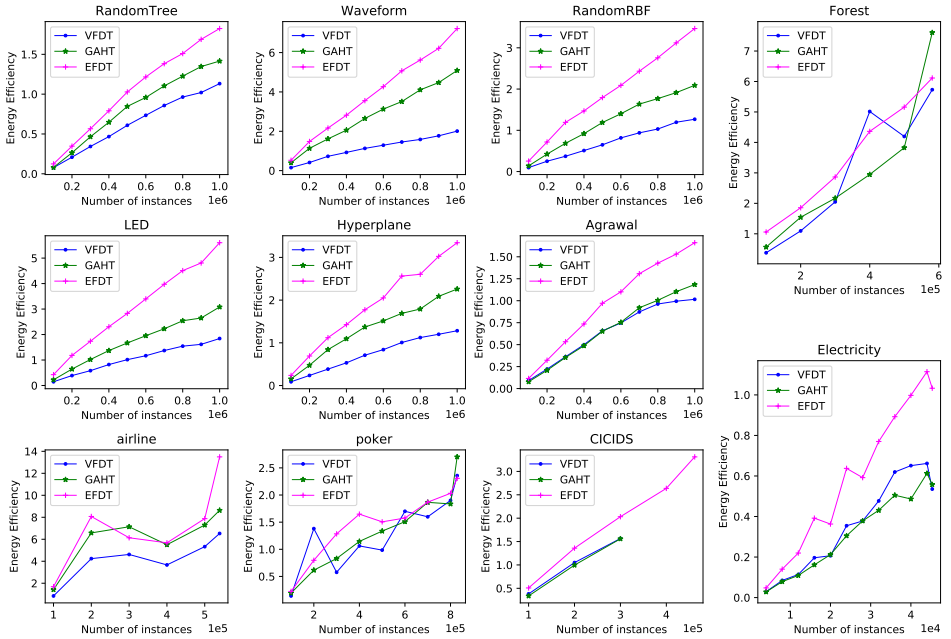

Figure 11.5: *Energy efficiency comparison between the VFDT, GAHT, and EFDT, for all datasets. Energy efficiency is calculated as* $\frac{energy}{kappa}$*, representing how much energy is needed to achieve a specific kappa statistic value. Kappa is the difference between the current accuracy and the accuracy obtained labeling the tree with the majority class. In this case the lower value of energy efficiency the better, showing that less energy is needed for that specific accuracy.*

11.7 Conclusions

This paper introduced the Green Accelerated Hoeffding Tree algorithm (GAHT), an approach to induce energy efficient Hoeffding trees with high accuracy levels. Our energy efficient approach is focused on building a tree adaptively in real time and per-branch, based on the data observed at each leaf. In particular, the algorithm deactivates the less visited leaves and grows faster the most visited leaves.

We compared GAHT against the EFDT (Extremely Fast Decision Tree), the most novel Hoeffding tree algorithm, on 11 publicly available datasets. The results show that we are able to achieve 0.16% higher accuracy on average, while reducing the energy consumption by 27%. The results also demonstrate that it is possible to grow smaller trees (compared to the

EFDT), while spending less energy, and still achieve competitive accuracy results. We introduced the concept of energy efficiency, as the amount of energy that is needed to achieve a specific accuracy. Ideally, the more energy that is spent, the higher the predictive accuracy. The results showed that the GAHT is more energy efficient than the EFDT on all datasets, and equally or more energy efficient than the VFDT on half of the datasets.

The results of this paper show that it is possible to advance data stream mining algorithms towards a more energy efficient design, while maintaining high accuracy levels. This contributes at making these algorithms more feasible to run in the edge, by smartly using the energy only on the necessary parts of the algorithm.

For future work we will test the GAHT on ensembles of Hoeffding trees, such as the Adaptive Random Forest algorithm, to reduce their energy consumption. We also plan to test the GAHT in other Hoeffding tree algorithms, such as the Hoeffding Adaptive Tree, to produce high accurate and energy efficient algorithms that are able to handle concept drift.

11.8 References

[1] P. Domingos and G. Hulten. "Mining high-speed data streams". In: *Proc. 6th SIGKDD Int'l Conf. on Knowledge discovery and data mining*. 2000, pp. 71–80.

[2] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet. "A survey on ensemble learning for data stream classification". In: *ACM Computing Surveys (CSUR)* 50.2 (2017), p. 23.

[3] A. Bifet, G. Holmes, and B. Pfahringer. "Leveraging bagging for evolving data streams". In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2010, pp. 135–150.

[4] C. Manapragada, G. I. Webb, and M. Salehi. "Extremely Fast Decision Tree". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '18. London, United Kingdom: ACM, 2018, pp. 1953–1962. ISBN: 978-1-4503-5552-0. DOI: 10.1145/3219819.3220005.

[5] W. Hoeffding. “Probability inequalities for sums of bounded random variables”. In: *Journal of the American statistical association* 58.301 (1963), pp. 13–30.

[6] J. G. Koomey, S. Berard, M. Sanchez, and H. Wong. “Assessing trends in the electrical efficiency of computation over time”. In: *IEEE Annals of the History of Computing* 17 (2009).

[7] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. Oord, S. Dieleman, and K. Kavukcuoglu. “Efficient Neural Audio Synthesis”. In: *International Conference on Machine Learning*. 2018, pp. 2415–2424.

[8] C. F. Rodrigues, G. Riley, and M. Luján. “Fine-grained energy profiling for deep convolutional neural networks on the Jetson TX1”. In: *Workload Characterization (IISWC), 2017 IEEE International Symposium on*. IEEE. 2017, pp. 114–115.

[9] C. F. Rodrigues, G. Riley, and M. Luján. “SyNERGY: An energy measurement and prediction framework for Convolutional Neural Networks on Jetson TX1”. In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2018, pp. 375–382.

[10] S. Han, H. Mao, and W. J. Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding”. In: *arXiv preprint arXiv:1510.00149* (2015).

[11] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks”. In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138.

[12] K. Gauen, R. Rangan, A. Mohan, Y.-H. Lu, W. Liu, and A. C. Berg. “Low-power image recognition challenge”. In: *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE. 2017, pp. 99–104.

[13] G. Hulten, L. Spencer, and P. Domingos. “Mining time-changing data streams”. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 97–106.

[14] A. Bifet and R. Gavaldà. "Adaptive learning from evolving data streams". In: *International Symposium on Intelligent Data Analysis*. Springer. 2009, pp. 249–260.

[15] R. Pelossof, M. Jones, I. Vovsha, and C. Rudin. "Online coordinate boosting". In: *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE. 2009, pp. 1354–1361.

[16] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem. "Adaptive random forests for evolving data stream classification". In: *Machine Learning* 106.9-10 (2017), pp. 1469–1495.

[17] N. Kourtellis, G. D. F. Morales, A. Bifet, and A. Murdopo. "VHT: Vertical Hoeffding Tree". In: *arXiv preprint arXiv:1607.08325* (2016).

[18] D. Marrón, E. Ayguadé, J. R. Herrero, J. Read, and A. Bifet. "Low-latency multi-threaded ensemble learning for dynamic big data streams". In: *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE. 2017, pp. 223–232.

[19] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, and V. Boeva. "Hoeffding Trees with nmin adaptation". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2018, pp. 70–79.

[20] V. Losing, H. Wersing, and B. Hammer. "Enhancing Very Fast Decision Trees with Local Split-Time Predictions". In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2018, pp. 287–296.

[21] A. Bifet, J. Zhang, W. Fan, C. He, J. Zhang, J. Qian, G. Holmes, and B. Pfahringer. "Extremely Fast Decision Tree Mining for Evolving Data Streams". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2017, pp. 1733–1742.

[22] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization." In: *ICISSP*. 2018, pp. 108–116.

[23] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. "RAPL: Memory power estimation and capping". In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. Aug. 2010, pp. 189–194. DOI: 10.1145/1840845.1840883.