Extraction and Energy Efficient Processing of Streaming Data

Eva García-Martín
“If I can’t dance, I don’t want to be part of your revolution.”

Emma Goldman
“Todo por El Humor”

Javier Espinosa
Abstract

The interest in machine learning algorithms is increasing, in parallel with the advancements in hardware and software required to mine large-scale datasets. Machine learning algorithms account for a significant amount of energy consumed in data centers, which impacts the global energy consumption. However, machine learning algorithms are optimized towards predictive performance and scalability. Algorithms with low energy consumption are necessary for embedded systems and other resource constrained devices; and desirable for platforms that require many computations, such as data centers. Data stream mining investigates how to process potentially infinite streams of data without the need to store all the data. This ability is particularly useful for companies that are generating data at a high rate, such as social networks.

This thesis investigates algorithms in the data stream mining domain from an energy efficiency perspective. The thesis comprises of two parts. The first part explores how to extract and analyze data from Twitter, with a pilot study that investigates a correlation between hashtags and followers. The second and main part investigates how energy is consumed and optimized in an online learning algorithm, suitable for data stream mining tasks.

The second part of the thesis focuses on analyzing, understanding, and reformulating the Very Fast Decision Tree (VFDT) algorithm, the original Hoeffding tree algorithm, into an energy efficient version. It presents three key contributions. First, it shows how energy varies in the VFDT from a high-level view by tuning different parameters. Second, it presents a methodology to identify energy bottlenecks in machine learning algorithms, by portraying the functions of the VFDT that consume the largest amount of energy. Third, it introduces dynamic parameter adaptation for Hoeffding trees, a method to dynamically adapt the parameters of Hoeffding trees to reduce their energy consumption. The results show an average energy reduction of 23% on the VFDT algorithm.

Keywords: machine learning, green computing, data mining, data stream mining, green machine learning
Preface

The author has been the main driver of all the publications were 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. The main supervisor provided expertise in machine learning. The co-supervisor provided expertise in computer architecture. Both supervisors contributed with comments and suggestions on conceived ideas, research designs, analyses of results, and paper drafts.

Included Papers


267-281), Springer. DOI: https://doi.org/10.1007/978-3-319-57186-7_21


**Related Papers**


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).
Acknowledgements

This thesis would have not been possible without the support of many people. First and foremost, I am deeply grateful to Niklas Lavesson, my main advisor, *el jefe de jefes*, and the reason why I am writing this today. He once convinced me to start this journey, and here I am three years later. Thanks jefe, it has been a pleasure, you never gave up on me. Your effort, persistence, and timeless advice is incalculable. I would also like to thank Håkan Grahn, my second advisor, for his advice throughout these years.

One is hardly working alone, and I definitely had great people around me every single day. I have the best colleagues and friends that one could ever dream of, loyal, funny, and sincere. Shahrooz, thanks for being my support during these years, thanks for making the office feel like "home away from home". Christian, Siva, there is nothing better than coming to work every day and having friends like you, that make you smile no matter what. Diego, thanks for that positive energy that you transmit every second of the day. Fredrik, thanks for your help and advice. Thomas, thanks for being the math guy. Veselka, the one and only boss, thank you so much for all those discussions, talks, and advice. You made a difference. Madrid is never forgotten, Javi, Amaia, Ane, Hen, keep fighting the good fight.

I am grateful to many colleagues from DIDD and other departments, thanks to those of you that tried to make a difference and created a positive and supportive environment, it does matter.

Finally but not less important, my family. I adore you, you are my pillars and always will be. Mom, dad, gracias. Elisa, mi amor, il mio amore, sei il mio sogno, grazie.

Karlskrona, November 2017
Eva García Martín
# Contents

Abstract ................................................................. i  
Preface ................................................................. iii  
Acknowledgements .................................................... v  

1 Introduction 1  
1.1 Research Problem .............................................. 2  
1.2 Contributions ................................................... 3  
1.3 Disposition ....................................................... 5  

2 Background 7  
2.1 Machine Learning ............................................... 7  
2.2 Data Stream Mining ............................................. 10  
2.3 Decision Trees .................................................. 12  
2.4 Online Decision Trees ......................................... 14  
2.5 Very Fast Decision Tree ....................................... 16  
2.6 Energy Consumption in Software ............................. 18  

3 Scientific Approach 21  
3.1 Research Questions ............................................. 22  
3.2 Research Method ................................................ 23  
3.3 Datasets .......................................................... 24  
3.4 Data Analysis .................................................... 26  
3.5 Energy Measurement .......................................... 27  
3.6 Validity Threats ................................................ 28  

4 Results 31
8.2 Background ............................................. 109
8.3 Energy Profiling of Decision Trees .................. 111
8.4 Experimental Design ................................. 114
8.5 Results and Analysis ................................. 117
8.6 Conclusions and Future Work ....................... 123
8.7 References ............................................. 123

9 Hoeffding Trees with $n_{\text{min}}$ adaptation 127

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

9.1 Introduction ............................................. 127
9.2 Background and Motivation .......................... 129
9.3 Methods and Technical Solutions ................... 133
9.4 Empirical Evaluation .................................. 137
9.5 Significance and Impact .............................. 143
9.6 References ............................................. 144
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, 3], image recognition at Facebook [4], and movie recommendations at Netflix [5]. Data stream mining is a subfield of machine learning that investigates how to process potentially infinite streams of data [6]. Data streams are usually infinite in length and change with time [7]. Thus, algorithms in this field have the ability to update the model with the arrival of new and evolving data, and by reading the data only once [7].

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) [8]. 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 [8]. One specific area is energy-efficient computing [9], where there is an important focus on reducing the energy consumption of data centers [10]. Energy efficiency has been important in the areas of computer engineering and computer architecture. For example, Intel processors have evolved to handle more operations by using the same amount of power [11]. Regarding machine learning, the interest on designing energy efficient algorithms is increasing, since the amount of computing intensive tasks, such as deep learning, is also increasing [12, 13].

This thesis explores green machine learning, which builds on green computing to design sustainable and energy efficient machine learning algorithms. In particular, we investigate energy consumption in data stream mining. While algorithms in this domain are known to consume small amounts of
1. Introduction

energy and memory space [14], they are designed to run constantly on data centers. Thus, reducing a small percentage of energy could have a significant impact at a larger scale. However, the energy consumed by stream mining algorithms is seldom evaluated [15]. Designing sustainable machine learning algorithms has an environmental impact, and allows for algorithms to run efficiently on embedded devices and battery powered devices. At the moment, both training and testing of convolutional neural networks is infeasible on mobile devices due to their high energy consumption [13].

The thesis includes two parts. The first part of the thesis presents a pilot study on data extraction and trend analysis, conducted with data from a streaming source, i.e. Twitter. The second and main part of the thesis investigates how to reduce the energy consumption of the Very Fast Decision Tree (VFDT) algorithm [14]. The VFDT is the first Hoeffding tree algorithm capable of analyzing data from a potentially infinite stream in constant time per example [14]. We follow three steps to achieve a lower energy consumption. First, we analyze how energy varies in the VFDT algorithm by tuning its parameters. Second, we identify the functions consuming the most amount of energy in the VFDT algorithm. Third, we present the dynamic parameter adaptation method suitable for VFDT and other Hoeffding tree algorithms. This method reduces the energy consumption of this class of algorithms by dynamically adapting the number of instances needed in a node for making a split decision (\(n_{min}\) parameter) based on the incoming data. The results show an energy reduction on the VFDT up to 87%, sacrificing at most 1% of accuracy.

1.1 Research Problem

The aim of this thesis is to explore efficient data analytics, with an emphasis on scalable and energy efficient solutions on large-scale datasets. In order to address such an aim, we focus on two objectives:

1. *Investigate how to extract and analyze data from large-scale datasets.*
   This objective is fulfilled in the first part of the thesis, with a pilot study that analyses data from a large-scale streaming source, Twitter.

2. *Investigate how to make Hoeffding tree algorithms more energy efficient.*
   This is the main focus of the thesis, to explore, with different levels
of maturity, how Hoeffding tree algorithms consume energy. It is addressed in the second part of the thesis. We first identify the energy bottlenecks of the Very Fast Decision Tree (VFDT) algorithm [14], the original Hoeffding tree algorithm, by showing which parameter setups and functions consume the highest amount of energy. We finally present dynamic parameter adaptation for Hoeffding tree algorithms, to trade-off energy efficiency against accuracy during runtime. To validate this approach, we introduce the nmin adaptation method in Hoeffding trees to reduce their energy consumption.

1.2 Contributions

The main contribution focuses on how to extract and process streaming data from an energy efficiency perspective. First, we conducted a pilot study on Twitter (Paper I), to understand how to extract, clean, and analyze data from a streaming source. Then, we investigated how to achieve energy efficiency in machine learning focusing on Hoeffding trees, and in particular focusing on the VFDT algorithm (papers II, III, and IV). Paper II gives a high-level overview of how energy is consumed by the VFDT algorithm. Paper III investigates and identifies the most energy-consuming functions of this algorithm. Finally, Paper IV presents a method to reduce the energy consumption of Hoeffding trees. Papers II, III, and IV have been conducted using large-scale real and artificial datasets. Figure 1.1 gives an overview of the aforementioned papers, and their relation to the aim and objectives described above. A more detailed summary and synthesis of these papers is presented below.

PAPER I Hashtags and followers: An experimental study of the online social network Twitter.

This paper studies the correlation between the use of hashtags and the increase of followers in Twitter. We do an exploratory analysis of a large user population in Twitter and investigate the characteristics of users that tweet with hashtags in comparison to users that do not tweet with hashtags. It is a pilot study into data analytics, where we extract, clean, and analyze high volumes of data originated from a streaming source.
1. Introduction

Figure 1.1: Papers included in the thesis. The thesis is divided in two parts. Each part addresses one objective, described in the two boxes below the aim.

PAPER II Energy Efficiency Analysis of the Very Fast Decision Tree algorithm.

This paper motivates the study of energy consumption in machine learning by analyzing the different energy consumption patterns of a well-known online decision tree algorithm. The results show that the algorithm consumes significantly different amounts of energy with different parameter values. This study shows that studying energy consumption is a challenging problem. These results motivated a further investigation to discover the energy hotspots (Paper III) and how to address them (Paper IV).


This paper identifies the energy hotspots of the Very Fast Decision Tree algorithm, the same algorithm studied in Paper II. Energy hotspots are the functions of the algorithm with the highest energy
1.3. Disposition

consumption. This study portrays: i) a methodology to measure energy consumption in decision trees, ii) an understanding of how and where energy is being consumed in the VFDT, and iii) suggestions of how to reduce that energy consumption. Paper IV addresses those energy hotspots.

**PAPER IV** **Hoeffding trees with \textit{nmin} adaptation.**

This paper addresses the energy hotspots identified in Paper III by presenting \textit{dynamic parameter adaptation} for Hoeffding trees, a method to reduce the energy consumption of this class of algorithms. To illustrate this method, we proposed the \textit{nmin adaptation} method to improve parameter adaptation in Hoeffding trees. This method dynamically adapts the number of instances needed to make a split (\textit{nmin} parameter) and thereby reduces the overall energy consumption. We applied this method in the VFDT, reducing its energy consumption by 23% on average while retaining accuracy.

**1.3 Disposition**

The remainder of the thesis is divided into four 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 the Very Fast Decision Tree. We conclude with a discussion about how energy is consumed by software.

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 the section above. Finally, Chapter 5 concludes with a summary and synthesis of the contributions and main points of the thesis.
2.1 Machine Learning

Machine learning addresses the question of how to construct computer programs that automatically improve with experience [16]. It has its foundations in artificial intelligence, computer science, and statistics. In 1950 [17], 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 [18]. In 1957, the first perceptron algorithm was created [19]. A few years later, machine learning started to gain importance after having a more specific focus on statistics.

Machine learning algorithms are divided into supervised, unsupervised, semi-supervised, and reinforcement learning. We mainly focus on supervised classification learning algorithms in this thesis.

Supervised Learning

Supervised learning is the machine learning task of learning a model that can map the labeled input data to the desired output [20]. This is formally defined as [20]:

\[
y = g(x|\theta)
\]

(2.1)

where \(g(\cdot)\) is the model, \(\theta\) are the parameters of the model, \(y\) is the output, and \(x\) is the input. The goal is to predict the output \(y\) based on new input data \(x\), given \(\theta\). In supervised learning, we can distinguish between
classification and regression. Classification are tasks where \( y \) takes class labels. On the other hand, regression tasks concern the prediction of a numerical output \( y \). For example, the price of a house, where \( y \) takes only numerical values.

The most simple type of classification is binary classification, where \( y \) can only have two different values. For example, a machine learning algorithm to decide whether a home is in San Francisco or in New York\(^1\), based on the elevation of the home and the price per sqm (square meter). This is a binary classification problem where [20]:

\[
x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\]  

(2.2)

\( x \), the input, has two attributes, \( x_1 \) and \( x_2 \). \( x_1 \) is the elevation, and \( x_2 \) the price per sqm. The classes are denoted by [20]:

\[
r = \begin{cases} 
1 & \text{if the house is in San Francisco} \\
0 & \text{if the house is in New York}
\end{cases}
\]  

(2.3)

Thus, each apartment is represented by an ordered pair \((x, r)\), and the training set \( \chi \) contains a sample of \( N \) instances of those pairs.

\[
\chi = \{ x^t, r^t \}_{t=1}^{N}
\]  

(2.4)

where \( t \) are the different instances in the set [20].

A more general type of classification is multi-class classification, where the algorithm can classify instances into more than two classes. Some algorithms with this ability are decision trees [21], multilayer perceptron [22], k-nearest neighbors [23], Naive Bayes [24], and support vector machines [25].

Regarding regression algorithms, they follow Eq. 2.1, where the goal is to predict an output \( y \) based on a model \( g(\cdot) \), and given some parameters \( \theta \). The most common regression algorithm is linear regression. The algorithm models the data based on a linear function of the form [20]:

\[
y = w_1 x + w_0
\]  

(2.5)

\(^1\)Example from: http://www.r2d3.us/visual-intro-to-machine-learning-part-1/
where the objective is to find the optimal weights $w_1, w_0$ that best fit the data. When a new data point arrives, we introduce the $x$ values in Eq. 2.5, and get a predicted value of $y$.

Machine learning algorithms can be organized and described in many ways. One way \[26\] is to divide algorithms into groups based on the models they generate. Following this criterion, they can be divided into geometric, probabilistic, and logical models.

Geometric models are created using geometric concepts such as planes, lines and distances \[26\]. The instances are visualized in a Cartesian instance space, where each feature represents a dimension. The main advantage is that the data points and the model are easy to visualize, as long as we consider a maximum of 3 dimensions. Examples of machine learning geometric algorithms are support vector machines and $k$-nearest neighbor. Support vector machines \[25\] are a class of algorithms that find the separating hyperplane between two classes. It was initially designed for binary classification problems, and was later extended to multi-class classification. The goal is to choose the hyperplane with the largest margin between both classes. This, in theory, would give a better classification accuracy for new instances. K-nearest neighbors \[23\] is a distance-based classifier that, based on the euclidean distance between each instance, classifies with the same class those instances that are close to each other.

Probabilistic models model the relationship between the output $y$ and the input $x$ based on a probability distribution \[26\]. This is the case of the Naive Bayes classifier \[24\]. It models the conditional probability distribution, $P(y|x)$, that is, given an input $x$ (features), it returns the probability distribution over $y$, the target or class. Bayes theorem is then used to calculate the conditional probability, which states that \[20\]:

$$P(y \mid x) = \frac{P(x \mid y) P(y)}{P(x)} \quad (2.6)$$

where $P(y)$ is the prior probability, and $P(x)$ is the probability of the data.

Logical models employ if-then rules built from logical conditions to determine homogeneous areas in an instance space \[26\]. Up to some extent, they can be interpretable, so that humans can understand the reasons behind the predictions. Examples of this class of algorithms are rule models and decision trees. Decision trees are a set of interpretable machine learning
2. Background

algorithms that represent the data in the form of a tree following a divide-and-conquer approach. A detailed explanation is given in Section 2.3.

Unsupervised learning

Unsupervised learning algorithms learn patterns from unlabeled data [26]. This is the case of clustering algorithms, which group data without previous information of the groups. Other types of unsupervised learning approaches include association rules [27] and matrix decomposition [28]. Association rules are patterns obtained from data that create associations between items that frequently occur together [26]. Matrix decomposition is used to discover hidden patterns in data by decomposing the original data into submatrices where each show a specific pattern [26]. Matrix decomposition is used in techniques such as principal component analysis (PCA) [29] for reducing the dimensionality of the features. It summarizes the existing features into new orthogonal features, called components.

2.2 Data Stream Mining

Data stream mining tackles the problem of analyzing and mining data from potentially infinite streams of data [6]. It addresses the challenges behind mining continuous flows of data that were generated in changing environments at a high speed [30]. To address these three properties, the data streams computational model proposes the following set of requirements [6, 31]:

1. Analyze one example at a time and mostly once, since with the high volumes and high speed of data it is unfeasible to have multiple passes.

2. Ability to incorporate new information and update the model at the speed that the data arrives.

3. Use a fixed amount of memory to avoid creating large models.

4. Detect changes and adapt the model to the current data.

Examples of data stream applications are social networks, internet of things (IoT) devices, mobile phones, and many other devices that are constantly generating data. During the past years, many data stream mining
2.2. Data Stream Mining

algorithms have been presented that can efficiently mine data from data streams for different tasks. The further subsections detail classification, regression and concept drift for data stream mining.

**Classification**

Classification is one of the most researched topics in data stream mining, since adapting traditional classification algorithms from data mining scenarios to data stream mining is non-trivial [6]. One of the key challenges is to choose a sample that can correctly summarize the stream, so that a decision based on that sample would be the same decision if we had seen the whole dataset. This is the key idea behind Hoeffding trees [14]. Hoeffding trees are decision trees that can analyze streams of data at constant time per example. They use the Hoeffding bound [32] to choose the correct sample size. Hoeffding trees are explained in Section 2.4. Many extensions have been made to the original Hoeffding tree algorithm, the Very Fast Decision Tree (VFDT) [14], to handle concept drift and adapt to changes in the input data. These extensions are explained in Section 2.4.

Apart from decision trees, rule models that also use the Hoeffding bound have been introduced, such as VFDR (Very Fast Decision Rules) [33]. Moreover, several publications have introduced nearest neighbor algorithms for data stream mining [34, 35]. The state-of-the-art algorithm can handle concept drift while self-adapting its memory consumption [36], very related to reducing the energy consumption approach that we focus on this thesis. Finally, Hulten and Domingos presented a new way to learn Bayesian networks from a streaming scenario [37].

**Regression**

Relevant work on regression has been conducted by Ikonomovska on regression trees [38]. This approach handles concept drift, contains perceptrons in the tree leaves to predict the best class at the leaf, and uses a binary tree to handle numeric attributes.

**Concept Drift**

Finally, many studies have focused on handling concept drift in data stream mining [39, 40]. If an algorithm is designed to create machine learning
models based on a stream, adapting to changes is a necessary requirement, since real world streaming datasets change over time. The first algorithm to handle concept drift was the CVFDT [41], by the same authors of the VFDT. This algorithm maintains a sliding window to check for the quality of old data. The next algorithm to handle concept drift was the UFFT [39], where the authors use the drift detection method (DDM) to detect concept drift, and prune the tree whenever concept drift is detected [42]. Moreover, an adaptive sliding window method, ADWIN [40], was introduced. This method dynamically adjusts the length of the window, removing the need for a fixed value. This method was later introduced in the Hoeffding Adaptive Tree (HAT), and Hoeffding Window Tree [43] algorithms.

2.3 Decision Trees

Decision trees are one type of machine learning algorithm that implements a hierarchical data structure in the form of a tree following a divide-and-conquer approach [20]. It is a nonparametric method that divides the input space into local regions, that are identified in a sequence of recursive splits in a smaller number of steps [20]. Each region represents the class to be predicted. A decision node implements a test function based on an attribute and the attribute values at the branches [20]. The algorithm recursively splits the node into one empty node for each branch. The empty node is then substituted by a leaf if the information observed at that node is homogeneous. The value of the leaf represents the class or target to be predicted. Otherwise, the empty node is replaced by a decision node based on the best attribute observed at that node. The recursion stops when the information at each node is homogeneous enough to be classified as one class. A tree can then be converted to a set of if-then rules by traversing the tree [20].

Figure 2.1 shows a standard decision tree, built using a toy dataset introduced by Quinlan [21], and used in many introductory books. In this figure, the root node is the attribute outlook, that separates the data between those instances where outlook=humidity, outlook=overcast, outlook=rain. All the nodes are the attributes of the dataset, and the leaves yes, no are the class to be predicted.

Decision trees can be used for solving different types of problems, such
as classification and regression. Regression trees build models where the target of the input dataset has a numerical value. Classification trees, such as the one in Figure 2.1, are used when the training dataset contains a nominal class as the target. Classification trees can handle both numerical and nominal attributes. The original decision tree algorithm is the ID3 [21]. The ID3 does the following: chooses the root node of the tree as the attribute with the highest information gain. Then it creates as many children as the number of attribute values and splits the data between those values. This process is repeated recursively until the data for the branch is homogeneous enough, thus it has all or most instances belonging to the same class. Many other decision trees after ID3 started to use a heuristic measure to calculate the best attribute to split on. The most common method is entropy, used also in ID3. Entropy is a concept introduced by Shannon [44], that measures the amount of information contained in a given message. It was used to understand what was the shortest possible message to send from a source to a receiver without losing information. More specifically to machine learning and decision trees, the entropy specifies the minimum number of bits needed to be able to classify a certain instance [16]. Given a collection $S$ of instances, the entropy of $S$ is [16]:

$$Entropy(S) = - \sum_{i=1}^{c} p_i \log_2 p_i$$
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 [16].

Regarding decision trees, if the dataset is partitioned in subsets based on a specific attribute, the goal is to see which partition gives the highest information. Information gain is the entropy caused by partitioning the examples according to that attribute [16], and is defined as [16]:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$ (2.8)

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 $A$ [16]. This equation represents that the information gain is the original entropy of the collection $S$, minus the entropy after partitioning $S$ with attribute $A$. The value of $Gain(S, A)$ is the amount of information saved when trying to predict a target value by using attribute $A$ [16]. In the case of decision trees, it is used to decide which is the best attribute to split on.

### 2.4 Online Decision Trees

Online decision trees are a type of decision tree used for data streams. These algorithms build a decision tree incrementally as the data arrives on
2.4. Online Decision Trees

constant time per example. Figure 2.2 shows a timeline of the different online decision tree algorithms and other related algorithms and techniques published recently. The first algorithm that could potentially handle infinite streams of data was the Very Fast Decision Tree (VFDT) [14]. The VFDT is an algorithm that was published in the year 2000, that is the baseline for many current online decision trees. A thorough explanation of this algorithm is given in Section 2.5. The main characteristics of the VFDT is that it can read an example from a stream in constant time, and thus scale with the number of instances. It obtains a high accuracy in comparison to many offline algorithms. However, it does not handle concept drift and numerical attributes. Concept drift refers to a change in the incoming data. State-of-the-art online decision trees are able to have an up-to-date tree that only maintains updated data and discards outdated data.

The first online decision tree algorithm to handle concept drift was the CVFDT [41], presented by the authors of the VFDT algorithm. The same year, the first algorithm to perform ensemble of decision trees on an online streaming setting was presented [45]. It is known as online bagging, an online way to apply sampling bootstrap. Two years later, Gama et al. [46] presented the VFDTC, an extension of the VFDT that could handle numerical attributes and that used Naive Bayes to label the leaves of the tree. In the standard VFDT, whenever the information is homogeneous enough on a leaf, the algorithm chooses the majority class to label the leaf (i.e. each new test instance will follow the path of the tree following the decision nodes until it reached a leaf, and will be classified based on the class of the leaf). VFDTC uses Naive Bayes instead of the majority class to determine the best class at the leaf. The same authors of the VFDTC presented the drift detection method (DDM) [42]. DDM is a method to detect concept drift based on the Poisson distribution, and how change in the data diverted from the original distribution of the incoming data.

The UFFT (Ultra Fast Forest of Trees) [47] algorithm was presented as an extension of VFDT using the numerical attributes handling feature and the Naive Bayes for leaf prediction from VFDTC. The UFFT creates a forest of binary trees for multi-class problems by creating a tree for each pair of classes. It can handle concept drift and uses a new splitting criteria at the node to choose the best split. On the same year, a hybrid method to choose between Naive Bayes or majority class to predict the class at
the leaf was presented [48]. Two years later, the same authors presented **Hoeffding Option Trees (HOT)** [49]. Option trees are considered as a middle ground between ensemble and single trees. This algorithm creates options nodes, to allow more than one path for each example so that the best path is chosen as the class in the test phase.

Finally, two Hoeffding Trees extensions where presented in 2009: **HAT** (Hoeffding Adaptive Trees) [43], and **ASHT** (Adaptive Size Hoeffding Trees) [50]. HAT uses the ADWIN algorithm [40] to adapt to concept drift, where a window of a certain number of instances maintains the statistics of the stream to detect for a change. The novelty of this approach is that it is parameter free, since the size of the window (often the most complicated parameter to set) will be adapted based on the Hoeffding bound. The authors of the HAT algorithm presented also the **ASHT** algorithm, a new bagging of online trees based on the ADWIN method [40].

### 2.5 Very Fast Decision Tree

VFDT is an online decision tree algorithm that builds a tree incrementally from data that originates from a stream, presented in Algorithm 1. The algorithm uses the Hoeffding bound [32], introduced in Eq. 2.9,

\[
\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{2.9}
\]

to ensure, with confidence \(1 - \delta\), that the split on a specific attribute would be the same if infinite number of instances would have been observed. This tries to build equally accurate decision trees as offline algorithms but analyzing just a portion of the data stream.

The process to build the decision tree by training on the data stream is the following: VFDT reads an instance, sorts the instance into the corresponding leaf (by following the decision nodes of the tree), and updates the statistics at that leaf. When there are sufficient statistics at a leaf, meaning that \(n_{min}\) instances have reached that leaf, the algorithm calculates the information gain for each attribute, and selects the two attributes with the highest information gain. If the difference between the information gain of the best two attributes (\(\Delta G\)), is higher than \(\epsilon (\Delta G > \epsilon)\), then there is a split on the best attribute. If that difference is smaller than \(\epsilon\), and smaller than
Algorithm 1: VFDT: Very Fast Decision Tree

1: \(HT\): Tree with a single leaf (the root)
2: \(X\): set of attributes
3: \(G(\cdot)\): split evaluation function
4: \(\tau\): hyperparameter set by the user
5: \textbf{while} stream is not empty \textbf{do}
6: \hspace{1em} Read instance \(I_i\)
7: \hspace{1em} Sort \(I_i\) to corresponding leaf \(l\) using \(HT\)
8: \hspace{1em} Update statistics at leaf \(l\)
9: \hspace{1em} Increment \(n_l\): instances seen at \(l\)
10: \hspace{1em} \textbf{if} \(n_{\text{min}} \leq n_l\) \textbf{then}
11: \hspace{2em} Compute \(\overline{G}_l(X_i)\) for each attribute \(X_i\)
12: \hspace{2em} \(X_a, X_b = \text{attributes with the highest } \overline{G}_l\)
13: \hspace{2em} \(\Delta G = G_l(X_a) - G_l(X_b)\)
14: \hspace{2em} Compute \(\epsilon\) using Eq. 2.9
15: \hspace{2em} \textbf{if} \((\Delta G > \epsilon)\) or \((\epsilon < \tau)\) \textbf{then}
16: \hspace{3em} Replace \(l\) with a node that splits on \(X_a\)
17: \hspace{3em} \textbf{for} each branch of the split \textbf{do}
18: \hspace{4em} Set new leaf \(l_m\) with initialized statistics
19: \hspace{2em} \textbf{else}
20: \hspace{3em} Disable attr \(\{X_p | (\overline{G}_l(X_p) - \overline{G}_l(X_a)) > \epsilon\}\)
21: \textbf{end while}

\(\tau\) \((\Delta G < \epsilon < \tau)\), then there is a split on the best attribute, since both attributes are equally good for a split. When there is a split, the leaf is substituted by a node, and the children are set with the attribute values. If none of those cases occur, then there is no split, and that leaf waits for more instances to make a confident split.

If \(n\) is the number of instances, and \(m\) are the number of attributes, the complexity is going to be based on these two variables. From the pseudocode of Algorithm 1, we observe that we loop over \(n\) iterations, since we iterate once for every number of instances. Sorting an instance to a leaf will depend on the depth of the tree, and that depends on the number of attributes. Based on the original implementation [14] of the VFDT, once an attribute is read, it is removed from that branch (but not for the other branches of the tree). Thus, the maximum depth of the tree is \(m\). The computational complexity of the rest of the algorithm (lines 1-20) is \(n/n_{\text{min}}\), since it will be
computed every $n_{min}$ instances. Thus, the total computational complexity is $O(n \cdot m) + O(n/n_{min} \cdot m)$, since $n >> n_{min}$, the time complexity of the VFDT is $O(n \cdot m)$.

### 2.6 Energy Consumption in Software

The aim of this section is to give a brief overview of how an algorithm consumes energy, and to explain the relationship between energy, time, and power consumption.

Power is defined as the rate at which energy is being consumed. The average power is defined as [51]:

$$P_{avg} = \frac{E}{T}$$  \hspace{1cm} (2.10)

where $E$ is the energy, measured in joules (J), $P_{avg}$ is the power measured in watts (W) and $T$ is the time interval measured in seconds (s). The instantaneous power $P(t)$ consumed or supplied by a circuit element is [51]:

$$P(t) = I(t) \cdot V(t)$$  \hspace{1cm} (2.11)

where $I(t)$ is the current, measured in amperes (A) and $V(t)$ is the voltage, measured in volts (V). The dynamic power, that is the total power dissipated in a circuit when the elements are active, is defined as [52]:

$$P_{dynamic} = \alpha \cdot C \cdot V_{dd}^2 \cdot f$$  \hspace{1cm} (2.12)

where $\alpha$ 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 [51]. To reduce the power consumption of processors, there are techniques such as dynamic frequency voltage scaling that reduces the clock frequency or the voltage that is used in modern processors [51].

Energy is defined as the effort to perform a task. It is the integral of power over a period of time [52]:

$$E = \int_0^T P(t) dt$$  \hspace{1cm} (2.13)
While power is the rate at which energy is consumed, energy is the amount of work done in a period of time.

Moving on to a more specific definition of energy consumption and execution time of a program, the total execution time of a program is [52]:

\[ T_{\text{exe}} = IC \times CPI \times T_c \quad (2.14) \]

where IC is the number of executed instructions, CPI (clock per instruction) is the average number of clock cycles to execute each instruction of the program, and \( T_c \) is the machine cycle time [52].

The total energy consumed by a program, E,

\[ E = IC \times CPI \times EPC \quad (2.15) \]

is the product of the IC, the CPI and the energy per clock cycle (EPC). EPC is defined as

\[ EPC \propto C \cdot V_{dd}^2 \quad (2.16) \]

Energy per instruction (EPI) is defined as the product between CPI and EPC, \( EPI = CPI \cdot EPC \) [52]. To reduce the energy consumption of a program, we can reduce the number of instructions (IC) or the EPI. This is non-trivial, since sometimes reducing the CPI (to reduce the EPI) can lead to having a higher IC, and vice versa. To reduce the EPI one can follow several approaches. One approach is to change instructions with a higher CPI with instructions with a lower CPI. For example, ALU instructions have a lower CPI than loading instructions, because loading requires accessing to memory, while ALU instructions do not. Another approach is to reduce the energy per clock cycle, with techniques such as dynamic voltage/frequency scaling (DVFS) [53]. We focus on reducing the number of instructions with a higher CPI, since it is a feasible approach from a software focus.
This thesis is connected to the areas of machine learning and computer engineering. More specifically, the thesis overlaps the areas of data stream mining and energy efficiency in software. Machine learning, a core sub-area of artificial intelligence, is a combination of statistics and computer science, where statisticians provide the mathematical framework of making inference from data, and computer scientists work on the efficient implementation of the inference methods [20]. The two foundational questions of machine learning are the following [54]: 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 the challenge of learning from streams of data. It appeared due to the progress in hardware technology that made possible for companies to store and generate large amounts of data [6]. This area addresses two questions [6]: i) How can one learn from data and process data in only one pass? and ii) How can one learn from evolving data? Computer architecture is an engineering or applied science discipline that focuses on designing a computer to maximize the performance while staying within cost, power, and availability constraints [55]. Nowadays, with the advancements of computer systems, one of the focuses lie on energy and power efficiency [52].

The foundational question of this thesis, that addresses the questions in all these areas, is: How can one construct computer systems that automatically learn from streaming data, in an energy efficient manner? To address this general question, this chapter presents the research methods used in the thesis to answer the specific research questions proposed in Section 3.1. We also detail the datasets, data analysis, energy measurement and validity threats.
3. Scientific Approach

3.1 Research Questions

RQ1. *Is there a correlation between the increase of followers and the use of hashtags in Twitter?*

Paper I addresses this question by investigating if users that tweeted with hashtags had a higher increase of followers in Twitter. Users tweeting with hashtags are more visible in Twitter, thus we wanted to investigate if the intuition that hashtags are correlated with the increase of followers was correct.

RQ2. *Does the energy consumed by the Very Fast Decision Tree vary when tuning the parameters of the algorithm?*

Paper II addresses this question by creating an experiment where the energy consumption and accuracy of the VFDT were evaluated under different scenarios. Energy consumption has hardly been evaluated in machine learning, and especially in data stream mining. We investigated if the energy consumption of the VFDT could be reduced, since that would have a significant impact in the global energy consumption.

RQ3. *What are the functions of the VFDT that consume the most amount of energy?*

Paper III addresses this question by presenting a methodology to calculate the energy consumption of the different functions of the VFDT. Once we showed that energy consumption was a relevant variable to study, we investigated where did the algorithm consume energy.

RQ4. *How can we reduce the energy consumption of the VFDT and other Hoeffding trees?*

Paper IV addresses this question by creating a method to reduce the energy consumption of Hoeffding trees. This method, called *dynamic parameter adaptation*, adapts the parameters of the VFDT during runtime and based on the incoming data. To conclude the thesis investigation, we showed how to reduce the energy consumption of Hoeffding trees.
3.2 Research Method

This thesis is based on quantitative methods. The thesis is divided in two parts, part one covers Paper I, and part two covers papers II-IV. Each part of the thesis follows a different research methodology. The experimental design for parts one and two are shown in Figures 3.1 and 3.2.

![Figure 3.1: Experimental design for the first part of the thesis. Paper I addresses this part.](image)

![Figure 3.2: Experimental design for the second part of the thesis. Papers II-IV address this part.](image)

We created a natural experiment to answer RQ1, that addresses the first part of the thesis. A natural experiment is an observational study where the subjects, in this case users, are exposed to the experimental and control conditions (use of hashtags) naturally, not by the choice or manipulation of the investigators [56]. This paper studies the effect of the use of hashtags on the increase of followers. The independent variable is the one that is causing the outcome, also called treatment [57]. For this study, the independent variable is the use of hashtags. The dependent variable is the one that depends on the independent variable, and is the outcome of the result of such influence [57]. In our study, the dependent variable is the increase of followers.

In order to study this effect, we designed an experiment with two groups of users sampled randomly from Twitter: an experimental group (users tweeting...
with hashtags), and a control group (users tweeting without hashtags). We gathered users for a complete week, and the information for each user was updated five times during one hour, to check for the increase of followers.

The second part of the thesis addresses RQ2-RQ4. All papers in this part follow a similar quantitative approach in the form of experiments. The experiments focused on studying the performance of a specific algorithm under different conditions, with different controllable and confounding variables [58]. More specifically, we measured the effect in terms of performance when varying the parameters of the Very Fast Decision Tree algorithm on different datasets. Performance in these studies is represented by the energy consumption and the predictive accuracy. Moreover, we measured this effect on specific functions of the VFDT (Paper III). Finally, we compared the standard VFDT against the VFDT with dynamic parameter adaptation (method introduced in Paper IV). Controllable variables were the choice of parameters and dataset. A confounding variable was the choice of tool to measure the energy, since it could affect the results without our control.

We designed an experiment where we measured the accuracy and energy consumption by running the algorithm several times on different scenarios and then averaging the results. Moreover, we conducted exploratory data analysis on the accuracy and energy results, to investigate relationships between different variables. In Paper II we investigated the trade-off between accuracy and energy, the relationship between the size of the tree, the accuracy, the power consumption and the execution time. Paper III investigated which were the functions consuming the highest amount of energy and the trade-off between energy and accuracy. Finally, Paper IV compared the accuracy and energy consumption for two variations of the VFDT algorithm. We also showed, through visualization, how the studied parameter varied in different datasets, to clarify the effect of the proposed method.

### 3.3 Datasets

The datasets for Paper I, and papers II-IV, have been obtained following a different procedure. In Paper I we created a script that extracted data from the Twitter API. We then cleaned that data, and created the set of users tweeting with and without hashtags. That process was followed by
a statistical analysis of the data and analysis of the results. The dataset contained 502,891 users, 252,957 tweeting without hashtags and 249,934 tweeting with hashtags. The information for each user (regarding the number of followers) was updated a total of 5 times in an hour span.

Papers II, III and IV follow similar approaches towards the gathering of the data. In contrast with Paper I, the data for Papers II-IV was already available and cleaned by other researchers. We used both real world and synthetic datasets that are commonly used by researchers in the field [59]. Table 3.1 shows the number of attributes, classes and types of attributes for each dataset used in papers II-IV. We used synthetic datasets to investigate the algorithm’s behavior under large datasets, since we can simulate almost an infinite data stream using a synthetic data generator. We used real world datasets to increase generalizability and to show that the methods presented in the studies also hold for real scenarios.

In particular, we used the following synthetic generators obtained from the massive online analysis (MOA) framework [31]: random tree, hyperplane, LED, and waveform [7]. The random tree generator builds a tree, inspired on the dataset proposed by the VFDT original authors [14], 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.

\[
\sum_{i=1}^{d} w_i x_i = w_0
\]  

(3.1)

where \( x_i \) is the coordinate for each point. More details are given in [41]. 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. There are a total of 24 attributes. Finally, the waveform dataset creates numerical values for a total of 21 attributes that represent the coordinates of three different types of waves.

The real world datasets used in papers II, III and IV are: poker, electricity and airline. 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 the electricity price based on different attributes. Finally, 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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nominal Atts</th>
<th>Numerical Atts</th>
<th>Classes</th>
<th>Used in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Tree</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>Papers II, III, IV</td>
</tr>
<tr>
<td>Hyperplane</td>
<td>0</td>
<td>10</td>
<td>2</td>
<td>Papers II, III, IV</td>
</tr>
<tr>
<td>LED</td>
<td>0</td>
<td>24</td>
<td>10</td>
<td>Paper III</td>
</tr>
<tr>
<td>Waveform</td>
<td>0</td>
<td>21</td>
<td>3</td>
<td>Papers III, IV</td>
</tr>
<tr>
<td>Poker Hand</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>Papers II, IV</td>
</tr>
<tr>
<td>Electricity</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>Paper IV</td>
</tr>
<tr>
<td>Airline</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>Papers II, IV</td>
</tr>
</tbody>
</table>

### 3.4 Data Analysis

We conducted several analyses on the data to answer the research questions from Section 3.1. Paper I investigates a correlation between hashtags and followers. We created two groups of users, one with hashtags and one without hashtags, and then evaluated the difference of the increase of followers between both groups. We first analyzed if the data followed a normal distribution with the Kolmogorov-Smirnov test [62]. Since the test indicated that the data did not follow a normal distribution, we conducted the non parametric Mann Whitney U test [63]. This test ranks the values of a control and experimental group and evaluates if there is a statistical difference between the ranks of both groups. The null hypothesis is the following: *is it equally likely that a random value from the control group will be less or greater than a random value from the experimental group?* The results of this test indicated that the users tweeting with hashtags had a significantly higher increase of followers than users tweeting without hashtags. We also conducted exploratory analysis of the collected data. We analyzed the presence of popular users, the tweet rate, and the URL presence. More details are given in Chapter 4.
Papers II-IV investigated the trade-off between accuracy and energy consumption for the VFDT on different scenarios. We evaluated the accuracy by calculating how many instances were correctly classified. Paper II also analyzed the relationship between the size of the tree (by measuring the number of nodes) with the accuracy, execution time, and power consumption. We visualized these relationships with several plots. Paper III analyzed the energy consumption and accuracy specifically for different functions of the VFDT. We averaged across all setups and datasets the most energy consuming functions, and compared them in different barplots. Paper IV compared how much energy can be reduced by using dynamic parameter adaptation. In the paper, we calculated the energy reduction and plot it for every value of \( nmin \) and for every dataset. We also compared the accuracy for all datasets between the two algorithm variants.

Regarding training and testing samples, papers II and III use the same number of instances for training and for testing. Paper IV uses 2/3 of the data for training, and 1/3 for testing. The energy consumption was calculated using different tools for each paper, more details are given in Section 3.5.

### 3.5 Energy Measurement

We used different publicly available tools to measure the energy for each study. Paper II uses the PowerAPI [64] tool to measure the energy consumption on the different scenarios. PowerAPI is a tool that uses different energy models to estimate the energy consumption based on the CPU utilization. The main disadvantage is that this measurement does not consider accesses to RAM and is mainly focused on the processor, so many details of how energy is consumed are missed. We then used the tool Jalen for Paper III, that uses the same models as PowerAPI, but is meant for java programs. The motivation to use this tool is that Jalen is able to output the energy consumption per function, exactly what we needed for our study. This allowed us to identify the energy bottlenecks of the VFDT algorithm. However, the same disadvantage occurs than with PowerAPI.

In Paper IV we used the Sniper simulator\(^1\) to address the disadvantages of having inaccurate energy models to estimate the energy consumption.

\(^1\)http://snipersim.org
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 RAM and the different caches. It gives a detailed view of the energy consumption. We can also inject \texttt{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 is very time consuming to simulate a simple algorithm run. That was the reason why we had to use small datasets to be able to simulate the VFDT under different scenarios. The same occurs when using \texttt{SimMarker()} functions, since analyzing the markers for all the function calls is also very time consuming.

3.6 Validity Threats

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

**Statistical Conclusion Validity**

Statistical conclusion validity addresses whether the correlation between two variables is true, and how strongly they are correlated [56]. The researcher can conclude that there is a correlation when there is not; or overestimate the magnitude of the correlation [56]. This validity threat applies only to Paper I, since we only conducted statistical tests in that paper. Paper I studies the correlation between hashtags and followers with a large sample of users, which increases the power of the test. Moreover, as explained in detail in the paper, we tested that the data was collected correctly.

**Internal Validity**

Internal validity refers to inferences about whether an observed correlation between groups reflects a casual relationship [56]. 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 analyzes a correlation between two variables. It also studies the presence of possible confounding variables (e.g. popular users, tweeting at a high rate, etc) to conclude that they are not affecting the increase of followers (dependent variable). Papers
II-IV do not measure a causation either. These studies measure the energy consumption by varying different parameters and datasets under the same circumstances (same algorithm, same computing platform, same tool).

**External Validity**

External validity addresses the generalizability of the results [67]. The data for Paper I was obtained only during a specific week, which impacts the generalizability of the results. However, we believe that external validity increases with the fact that we collected users for every day and for every hour of the day.

Papers II-IV are concerned with a specific implementation of a specific algorithm. The results can not be generalizable to other algorithms and other implementations. To mitigate this effect, we tested a different implementation of the same algorithm and obtained similar results. This, together with a theoretical analysis of how the algorithm consumes energy, increased the external validity.

**Construct Validity**

Construct validity addresses if a test measures what it claims [56]. 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, hashtag, followers, energy consumption and accuracy. These variables have already been defined throughout the text.
This chapter first presents the answers to the research questions in Section 4.1, and then details the general and specific contributions of the thesis in Section 4.2.

4.1 Answers to RQ1-RQ4

RQ1. *Is there a correlation between the increase of followers and the use of hashtags in Twitter?*

The results of the first study, Paper I, showed that there is a correlation between the use of hashtags and the increase of followers. This was obtained from the Mann Whitney U test. This test indicated that there was a significant difference between users tweeting with hashtags and users tweeting without hashtags. In particular, the average increase of followers for users tweeting with hashtags was 2.7 times higher than for users tweeting without hashtags.

RQ2. *Does the energy consumed by the Very Fast Decision Tree vary when tuning the parameters of the algorithm?*

The results of Paper II showed that there is a difference in energy consumption when varying the parameters of the VFDT, with a 90% energy increase for some setups. This illustrated the impact of introducing energy consumption in a traditional data stream mining setup.

RQ3. *What are the functions of the VFDT that consume the most amount of energy?*

Paper III showed that, on average, the functions related to labeling the leaves of the tree with a specific class were the most energy-consuming functions.
4. Results

consuming. The reason for this was that the default parameter setup of the algorithm used both Naive Bayes and majority class as methods to choose the best class. That doubled the computation compared to choosing one of both methods. Moreover, sorting the instance to the leaf, handling numeric attributes, and calculating the best attributes were also high energy consuming functions. More details are given in Section 4.2.

RQ4. How can we reduce the energy consumption of the VFDT and other Hoeffding trees?

To address the energy hotspots discovered in Paper III, Paper IV presents dynamic parameter adaptation, a method to reduce the energy consumption of the VFDT and other Hoeffding tree algorithms. In particular, this paper introduced $n_{min}$ adaptation, a method that dynamically adapts the $n_{min}$ parameter of Hoeffding trees based on the incoming data.

4.2 Contributions of Papers I-IV

This thesis presents two mayor contributions related to both parts of the thesis. The first contribution shows how to extract, clean, explore, and analyze data from a streaming source, Twitter, as a pilot study and first step into processing and mining streaming data. The second and main contribution shows how to achieve energy efficiency (in the means of reducing the energy consumption) for a particular class of decision trees in data stream mining. The following paragraphs address the specific contributions of each paper.

Paper I

Paper I, apart from presenting the correlation between hashtags and the increase of followers, conducts an exploratory analysis of the data gathered from Twitter. We observed that users tweeting with hashtags lost more followers than users tweeting without hashtags. This suggested that users tweeting with hashtags are more visible in Twitter, which is understandable since users can search tweets based on a specific hashtag.

Furthermore, tweets with hashtags contained more URLs, observation that was validated with another study in the same field [68]. We also
investigated the presence of popular users in both the hashtag and non-hashtag group. Popular users are those users with a high number of followers. We observed that there are more popular users tweeting with hashtags than without hashtags. However, the results suggested that there was no apparent relationship between popular users and the increase of followers, with regards to the hashtag usage.

Finally, we investigated the increase of tweets for users tweeting with and without hashtags. We controlled for the increase of tweets as a possible reason for the increase of followers. We did this by calculating how many users with a high increase of followers were tweeting at a high rate, from the hashtag and non-hashtag group. The results showed that the hashtag and non-hashtag group had similar amount of users that had both a high increase of followers and a high increase of tweets (only 2% difference), concluding than a higher increase of tweets on the hashtag group was not the reason for the higher increase of followers.

**Paper II**

Paper II, apart from evaluating the energy consumption for different parameter setups, evaluated the trade-off between energy consumption and accuracy. The results showed that there was no apparent trade-off between accuracy and energy consumption. For two datasets a higher accuracy required a higher energy consumption on average. However, for the other two datasets, a higher accuracy required lower energy consumption, which showed very promising results and motivated future studies.

Moreover, we investigated the relationship between accuracy, energy and the size of the tree. The results showed that, on average, the bigger the tree, the higher the accuracy and the higher the energy consumption. A tree with more nodes is able to better represent the relationship between the attributes, but more energy is needed because there are more operations involved. However, Paper IV showed that very few splits can result in a higher energy consumption, especially if the attributes are numerical, since more memory is allocated to save the information at each node.

These results motivated us to study where exactly in the algorithm the energy is being consumed, and how to reduce the energy consumption by modifying some parts of the algorithm.
4. Results

Paper III

Paper III contributed also with a detailed explanation of why certain functions are consuming more energy than others in the VFDT. The VFDT can store the statistics of nominal instances at a leaf by keeping a table with counters for each attribute value. However, updating the statistics for numerical instances is more computationally inefficient. Since, depending on the method, the leaf needs to store many instances [39], or compute several operations for each instance update [7, 69]. Calculating the best attributes is an expensive operation since the algorithm has to calculate the information gain from all attributes observed at each node. It is governed by the $nmin$ parameter, that decides how often this function is called.

Paper III contributed also with a methodology to identify energy hotspots of decision trees. The most energy consuming functions were identified and mapped to generic functions of the same algorithm class. Mapping specific functions of an algorithm to generic functions allows for a better understanding of where energy is consumed, and allows for further comparison with other algorithms that share the same generic functions.

Paper IV

Paper IV addressed the energy hotspot identified in Paper III related to calculating the best attributes. In this paper we looked at a different implementation of the VFDT, and evaluated which functions were consuming the highest amount of energy. We also used a more precise tool to measure the energy. The results of the analyses showed that, similarly to Paper III, calculating the best attributes and handling numerical attributes were consuming significant amount of energy. After this investigation, we discovered that if the algorithm calculated the best attributes only when there would be a split, that would reduce the energy consumption significantly.

To solve this problem, we introduced the $nmin$ adaptation method (already explained). This method adapts the value of $nmin$ to a value that guarantees a split whenever the best attributes are calculated (the function calculating best attributes is called based on the parameter $nmin$). The results showed that this method reduced the energy consumption by a 23% in average, comparing VFDT with VFDT with $nmin$ adaptation. However, accuracy was affected less than 1% in average.
Conclusions and Future Work

This thesis comprises four research papers, that address the two objectives of the thesis. The first objective focused on how to extract and analyze data from large-scale datasets. We addressed this objective by presenting a study (Paper I) that shows how to extract data from a streaming source, and how to create a dataset for further study. This was a pilot study and an introduction into streaming data.

The second objective focused on how to make Hoeffding tree algorithms, a class of data stream mining algorithms, more energy efficient. Papers II-IV addressed this objective in different steps. First, we evaluated if there was a significant difference in energy consumption when varying the parameters of the VFDT algorithm. Energy consumption had not been evaluated before for this class of algorithms. Thus, we believed that it was necessary to investigate if energy consumption was a relevant variable to study. The results indicated that there were different energy consumption patterns for different parameter setups, showing the relevancy of evaluating energy consumption. Based on these results, we investigated further to discover where exactly in the algorithm the energy was being consumed. The goal was to portray the most energy consumption functions, i.e. energy hotspots, and reduce their energy consumption. Finally, once the energy hotspots were discovered, we presented a method to reduce the energy consumption of Hoeffding trees. The results of this study were very promising, since they showed an average reduction of energy of the 23% without affecting accuracy.

Reducing the energy consumption of algorithms is necessary for running algorithms in embedded systems, and desirable for data centers. Data stream mining algorithms are designed to run in servers continuously without much interruption. That is the reason why this thesis focuses on this class of
algorithms, since reducing their energy consumption can have a significant impact in the long term. This thesis presents the first approach to make data stream mining algorithms more energy efficient.

Many future directions can be taken, since we are proposing a novel approach to design machine learning algorithms. In the short term, we plan to test the method proposed in Paper IV in different Hoeffding tree algorithms and evaluate their energy consumption. This would be particularly useful for ensembles of Hoeffding trees, since the energy consumption is significantly higher than for the standard VFDT. In the long term, we plan to introduce the notion of green machine learning in different application domains, to encourage companies and other researchers to build machine learning algorithms from an energy efficiency perspective. Another long term goal is to evaluate the energy savings of introducing this and other techniques on already deployed systems in companies.


[63] H. B. Mann and D. R. Whitney. “On a test of whether one of two random variables is stochastically larger than the other”. In: *The annals of mathematical statistics* 18.1 (1947), pp. 50–60.


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.
Hashtags and followers: An experimental study of the online social network Twitter

Eva García-Martín, Niklas Lavesson, Mina Doroud

Abstract

We have conducted an analysis of data from 502,891 Twitter users and focused on investigating the potential correlation between hashtags and the increase of followers to determine whether the addition of hashtags to tweets produces new followers. We have designed an experiment with two groups of users: one tweeting with random hashtags and one tweeting without hashtags. The results showed that there is a correlation between hashtags and followers: on average, users tweeting with hashtags increased their followers by 2.88, while users tweeting without hashtags increased 0.88 followers. We present a simple, reproducible approach to extract and analyze Twitter user data for this and similar purposes.

6.1 Introduction

Twitter is a social network founded in 2006 that publishes around 1 billion tweets every two days [1], with the goal of spreading world breaking news or opinions about certain events [2]. The research conducted on Twitter has been centered in sentiment analysis, event prediction, retweet prediction and graph modeling.

Twitter has a key characteristic that was later adopted by other social networks, the hashtag symbol, commonly used to tag posts into different categories. Although there has been a lot of research conducted in Twitter in the past years, we believe that not enough studies have been published that try to discover how hashtags affect users in Twitter. Hashtags have gained relevance in the past years since companies use them to marketize events, tv shows and sport matches, in order to attract users engagement towards their
brand. A way to increase this engagement is by gaining followers on Twitter on the companies account. It has to be noted that having more followers do not always mean being more influential. The reason is that retweets and mentions are not strongly correlated with the number of followers, being retweets and mentions two measures that capture the real audience reactions. This means that follower count is not enough to capture the influence of a user, since it only measures the size of the audience, not its reaction [3].

For these reasons and considering that there is still a need to understand how users react to posts with hashtags, we have conducted a large-scale exploration of the potential correlation between the use of hashtags and the increase of followers on Twitter. More specifically, a natural experiment has been created where the average increase of followers is compared between two groups of distinctive users, one tweeting with hashtags and another tweeting without hashtags. The results of the experiment show that users tweeting with hashtags gain more followers than users tweeting without hashtags. To avoid confounding factors on the results, users tweeting with and without hashtags have been picked following the exact same random procedure, ending up with a random Twitter-user population. Therefore, if confounding variables appear, they would be canceled out [4].

Some analyses have been made in relation to popular users. Due to the fact that Twitter considers popular tweets as those that generate more engagements, we believe that users with a high number of followers could be considered popular users, since they have more probability to create engagement towards them. Our results show that popular users that tweet with hashtags gain more followers than popular users tweeting without hashtags. The main reason behind this is that the hashtag channel is attracting more followers to the already popular users. When a user clicks on a hashtags, popular tweets will appear first, thus increasing the probability for popular users to gain visibility.

Although there is a clear relationship between the use of hashtags and the increase of followers, this study does not go beyond than portraying a correlation between both variables. A cause-effect claim can not be made between using hashtags and increasing followers. However, we can recommend the use of hashtags to increase visibility in Twitter, as the results suggest.

The article is structured as follows: In Section 6.2 the background is presented with the Twitter-related terminology and the different articles
related to social networks. In Section 6.3 the purpose statement is detailed. In Section 6.4 the research methodology, the experimental design, data collection and statistical analysis are described. In Section 6.5 the raw and analyzed results are presented. In Section 6.6 we summarize the conclusions. Finally, in Section 6.7 we provide further perspectives of future work.

6.2 Background

6.2.1 Terminology

Tweets are short 140 character messages. To tweet is the action of posting a tweet in Twitter. Twitter users tweet to show to their followers their thoughts about a specific matter, to post breaking news or to post information about topics they are interested in [5]. In the following paragraphs we are going to detail all the Twitter glossary \(^1\). Mentions are used for connecting users. If user \(U_a\) wants to mention user \(U_b\), \(U_a\) posts the character @ followed by \(U_b\) username, e.g. @username. Retweet is a particular case of mentioning. A retweet is the action of a user tweeting the tweet of another user. Retweets are either created automatically, by clicking on the retweet button; or manually, by placing the text indicator \(RT\). If the user has manually retweeted a message with modifications, we use the text indicator \(MT\) \(^2\).

When users want to categorize their messages into specific topics, they add the hash symbol to the topic. For example \(#computerscience\). The hash symbol plus the topic or word is called a hashtag \(^3\).

Users connect with each other by following each other. If \(U_a\) follows \(U_b\), then \(U_a\) receives in his or her timeline all the tweets from \(U_b\). In this case, \(U_a\) is a follower of \(U_b\) and \(U_b\) is a friend of \(U_a\).

For gathering data from Twitter, there is an open API available for developers\(^3\). This API makes it easy for the developer to send requests to Twitter to ask for specific information [6]. In Section 6.4.2 we detail how the data was gathered from the API. In order to allow Twitter to monitor the number of requests we make, we need to follow an authentication protocol, \(Oauth\) \(^4\). The information is then gathered by the authenticated user.

\(^1\)https://support.twitter.com/articles/166337-the-twitter-glossary  
\(^2\)https://support.twitter.com/articles/166337  
\(^3\)https://dev.twitter.com/  
\(^4\)https://dev.twitter.com/docs/auth/oauth
6.2.2 Related Work

This section is organized in three parts. First, we briefly introduce studies related to online social networks in general. Second, we detail the three main focuses, mostly in Twitter, more closely related to this study. Finally, we end up detailing the two closest works to this one and explaining the main differences between them.

Online social networks are becoming more and more popular nowadays. Several models have been built that try to define the behavior of such networks [7, 8]. The most common model is the preferential attachment, created by Barabási et al. [9] and tested with positive results by Newman [10] and Jeong et al. [11]. Preferential attachment states that new links tend to form towards already popular links. Popular links are users that have a higher number of followers compared to the average Twitter user. On the other hand, Lang et al. [12] built a growth model of the social network Buzznet looking for evidence of preferential attachment. They unexpectedly discovered that Buzznet follows an anti-preferential attachment model; therefore, high-degree nodes create edges to low-degree nodes. Degree can be defined as the number of connections that a node creates towards other nodes. This means that users that we expect to have a higher number of followers than friends (high-degree nodes), such as celebrities, end up having a lower number of followers than friends (low-degree nodes).

Moreover, several studies such as Sakaki et al., Ritterman et al., Qiu et al. [13–15] have been made on event prediction using Twitter as a source. Based on hashtags or trends they try to predict the appearance of future events. We have also observed that social networks have been used by many researchers to detect the sentiment from different users to certain products or events. This is called sentiment analysis and it used both to analyze users opinions individually and to analyze opinions of groups of users. Looking into users as a group creates a more complete solution when combined with traditional sentiment analysis approaches. [2, 16–21].

In relation to the research conducted in Twitter, we have identified three main focuses related to this study. The first, influence, tries to identify the reasons behind influential posts on Twitter. Influence stands for "the power to cause an effect", in this case, the power to cause users to take certain actions, for instance retweet a certain tweet. Originating from the traditional influential theories, it was widely believed that only a minority group of
people, influencers, were capable of causing influence to other users [22, 23]. However, it was later discovered that ordinary users can also create influence, and that influence depends more on the type of posts, how society reacts to a specific trend and how close are the opinions of such users [3, 24]. The discovery most related to this study, is that indegree, that is, the number of followers of a user, is not a strong measure of the influence of such user, however retweets and mentions are [3]. A final remark from the same authors, is that users with active followers are more likely to be retweeted, and since retweet is an important measure of influence, we could conclude that having active followers increases the influence of a user.

The second focus is related to link prediction. Link prediction studies different factors that are related to a user gaining followers. A first study was conducted where researches discovered that links in Flickr were usually created by users who already had many links [8]. In the Twitter area, it has been shown that the topic of the tweet is one of the indicators to predict whether users will gain or loose followers. Tweets with happy messages gain followers in contrast to tweet with sad messages [25, 26]. Another important factor, stated by Hutto et al. [27], is the shared interests between the users. As was noted in the influence aspect, there are more links between users that have similar interests. On the same line, popular users often remain popular, in contrast to ordinary users. Regarding the friend and follower network [28], several studies focus on modeling this network into balanced or unbalanced, depending if two users have a friend in common whom they both follow, and its impact into maintaining followers in Twitter [29]. Other studies focus on predicting future friend nodes in the network [30].

The third focus is on the hashtag topic. A discovery was made by Suh et al. [31], where they state that tweets containing hashtags and URLs are more likely to be retweeted. They also mention that the amount of followers of a user does affect retweetability. This indirectly contradicts the claims by Cha et al. [3], since they mention that influence is not related to the amount of followers but that retweets are. However, if followers and retweets are related, based on the claim by Suh et al. [31], then followers should also be related to influence. In relation to the hashtag topic, Kong et al. [32] have created a study where they predict trending topic hashtags in real-time. Since the topic of the tweet is related to increase of followers, as was stated before, we believe that analyzing which hashtags attract more followers is a perfect follow up work. Regarding prediction, Maruf et al. [33] predicts the
personality of a user analyzing their hashtags and She et al., Otsuka et al., Yu et al., Y. Wang et al. [34–37] have created several recommender systems that suggest hashtags to the Twitter user.

Finally, Hutto et al. [27] and [38] have created two studies related to ours. [38] extract that users in Instagram do attract more followers and likes if such users adds hashtags to their publications. Hutto et al. [27] have conducted a complete study on the reasons behind follower growth patterns. They measure the relationship between many control variables and the increase or decrease of followers. They conclude that hashtags is one of the factors related to the increase of followers, matching with our discovery. The new perspective of our study is the ability to isolate hashtag usage from all other variables to ensure that the measured correlation is not affected by any other variable. Our complete experimental study is designed to study the correlation between hashtags and followers and avoid any confounding variables. Hutto et al. [27] study is more a general analysis into active users.

6.3 Purpose Statement

The purpose of this experimental study is to investigate the potential correlation between hashtag usage and the increase or decrease of followers; controlling for retweets, user characteristics and user popularity. The independent variable is hashtags. The dependent variable is the change in the number of followers. In this natural experiment we control for retweets, user popularity, user characteristics, the million follower fallacy [3] and new mentions. We take into account the effect of retweets since one of the key reasons for new followers is that $U_a$ starts to follow $U_b$ because they had a friend in common that retweeted a tweet from $U_b$.

The million follower fallacy is a term used when $U_a$ starts following $U_b$ just for etiquette or for being polite to follow someone that already follows you. As for user characteristics, we sample random users to have a set of users that is representative of the whole population, with random ages, genders, nationalities, languages and popularity level. Popular users are users widely known to the public with many links towards them, such as celebrities and famous sport players. We also consider the fact that $U_a$ mentions $U_b$ in his or her tweet. The username of $U_b$ is publicly being seen by all the followers of $U_a$, increasing the chance of $U_b$ to get new followers.

Finally, as for confounding variables, we consider the following three:
• A user posting his or her user account on a public place in the Internet.
• Twitter suggesting to follow new users.
• A Twitter user tweeting with a specific hashtag when attending a concert, conference or other events that might attract new followers.

We are aware that other confounding variables might appear in the future. However, in relation to the third bullet point, we are not studying the type of hashtag that leads to an increase of followers, we are studying if the presence of any hashtag affects this increase.

At this point, we have disclosed which control and confounding variables we are going to take into consideration. The way these variables are going to affect the experiment design and how we are going to control them is explained in Section 6.4.

6.4 Research Methodology

In order to investigate the possible relationship between the use of hashtags and the increase of followers, we propose a specific research question, namely; whether the addition of hashtags to tweets produces new followers.

We want to discover if hashtags are related to user visibility. In Twitter, if you click on any hashtag, trendy or not, you can see the users that are tweeting with that hashtag. Hence, we want to discover if users that search tweets based on a specific hashtag, actually start to follow the authors of those tweets. For that reason, we hypothesize that there is an increase in the number of followers for users tweeting with hashtags.

6.4.1 Experimental design

The aim is to perform a natural experiment to determine if hashtags are related to the increase of followers. The main characteristics that we want to achieve are randomization, non biased choices and a clear distinction between users tweeting with hashtags and without hashtags.

We created two independent groups of users, an experimental group and a control group. All users from both groups are gathered following the same procedure, being the only difference between them the hashtag usage. Therefore, since the sample group is large enough, if any confounding
variables emerge, they will appear in both groups and they will be canceled out. This is known as *random assignment* [4], where the objective is to minimize the chance that a difference between both groups will be due to confounding factors. Users that have tweeted with a hashtag in the moment they were collected form part of the experimental group and users that have tweeted without a hashtag form part of the control group.

There is also a relevant design choice made, being, that tweets from the past are not considered. The main reason is that it is not possible to know if past tweets attracted new followers, since we can only know the number of followers in present time. We are aware that other factors might still influence the results, therefore, we plan to identify and address such factors in subsequent work. With this experimental design and after a statistical analysis of the data we are able to answer the research question.

### 6.4.2 Data collection

Tweets and users are obtained from Twitter using the Twitter API. We developed a Python script to make the different requests to the Twitter API, the code is described in section 6.4.3. We used an open source package called Twython, from developer Ryan McGrath\(^5\), to make the different requests to Twitter.

From the Twitter API two requests were used:

- GET statuses/sample
- GET users/lookup

The first request returns a small sample of random tweets from Twitter’s public timeline independent of the location or language. For every tweet that does not contain a hashtag, its author becomes a member of the control group, and for every tweet that contains a hashtag, its author becomes a member of the experimental group. The following information is saved for each user: *Username*, *number of tweets*, *number of followers*, *number of friends*, *gathering date*, *number of times that tweet has been chosen as favorite*, *number of times that tweet has been retweeted*, *the tweet text*.

The second request returns the information from a maximum of 100 users. We use this request to update the information of each user to discover

\(^5\)https://github.com/ryanmcgrath/twython
6.4. Research Methodology

if the number of followers increased or decreased since they were collected. Every user is updated a total of 5 times, every 12 minutes. The reason we choose to update every 12 minutes in an hour span, is that based on a study conducted by Lardinois [39], the lifespan of a tweet is roughly one hour. Therefore, it is more likely that if an increase of followers occurs after one hour, it is due to another tweet or external factor. Every update is made after 12 minutes to see which of the 5 updates have the highest increase of followers; to match this with the study by Bray [40] which states that a tweet is more popular during the first 18 minutes after it was published.

There were a total of 502,891 users acquired, 252,957 users tweeting without hashtags and 249,934 tweeting with hashtags. There are some special considerations relevant for this study. Twitter API has a limit in the total number of requests that the developer can make per hour, therefore we have gathered the total number of users based on this limit. The updates of all the users could not be made a total of 5 times in all cases, because on rare occasions Twitter’s service was unavailable due to the server being overloaded with requests. To ensure that we make the correct calculations, we have removed all users that were not updated 5 times and whose updates where outside the interval between 1 and 2 hours.

6.4.3 Python Script

Script 2 portrays the code for obtaining all users and tweets from Twitter\(^6\).

```
Algorithm 2 To obtain a sample of tweets from Twitter with Twython.
1: users = []
2: days = 7
3: TOT-HOURS = 24 * days
4: for hour in range (TOT-HOURS) do
5:   tweets = TwythonStreamer
6:   users = getUsers(tweets)
7:   saveTweets(tweets)
8:   saveUsers(users)
9:   for update in range (5) do
10:     sleep(12)
11:     udpt-users = twitter.LookupUser(users)
12:     db.users.update(udpt-users)
```

\(^6\)https://github.com/evek2/tw-hashtags
The first step is to obtain the random sample of tweets from Twitter with the functions already implemented in Twython. We save these tweets as a collection in MongoDB named tweets. The next step is to extract the authors of such tweets and save them in another collection on MongoDB named users.

Moreover, after the users have been obtained, we update their information (number of tweets, number of followers and number of friends) five times in one hour, once every 12 minutes. We append this new information to each document from each user created in the second step. When the five updates are finished, new users are gathered and the process is repeated again for a complete week. In summary, we have a script that is going to be executed during one week, picking up new users every hour and updating their information every 12 minutes. Sometimes this interval will be higher due to Twitter rate limits.

The next step is to preprocess the data to calculate the difference between the number of followers between the last and the first instance. Finally, we import the data into a statistical tool (Matlab in this case) to perform the statistical analysis.

### 6.4.4 Data analysis

We first perform the Kolmogorov-Smirnov normality test [41] on the difference of followers for the control and the experimental group. In Figure 6.1 we represent the histogram of such differences, having zoomed both axis for an easier display. The results of the normality test are shown in Table 6.1.

The null hypotheses are the following:

- “The values that measure the difference in the number of followers for the control group follow a normal distribution”.
- “The values that measure the difference in the number of followers for the experimental group follow a normal distribution”.

Both hypotheses are rejected, since the p-value is less than 0.05, concluding that the difference of followers for both groups does not follow a normal distribution.

Since the data is tested negative for normality, we are going to perform a non-parametric test, namely, the Mann-Whitney U-test [42].
Figure 6.1: \textit{Increase of followers}. Approximation of the probability distribution of the increase of followers for the users who tweet with and without hashtags. Users tweeting with hashtags (blue) have a higher increase of followers than users tweeting without hashtags (orange).

<table>
<thead>
<tr>
<th></th>
<th>Control</th>
<th>Experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-value</td>
<td>&lt; 0.05</td>
<td>&lt; 0.05</td>
</tr>
<tr>
<td>Confidence level</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>Final result</td>
<td>Hypothesis rejected: The data does not follow a normal distribution</td>
<td>Hypothesis rejected: The data does not follow a normal distribution</td>
</tr>
</tbody>
</table>

The null hypothesis is: “The mean of the populations of the experimental group and the control group are equal”.

The alternative hypothesis is the following: “The mean of the populations of the experimental group is significantly higher than the mean of the populations of the control group”. In this test, the absolute value of parameter $z$, 

57
6. Hashtags and followers: An experimental study of the online social network Twitter

\[ z = \frac{U - \frac{n_1n_2}{2}}{\sqrt{\frac{n_1n_2(n_1+n_2+1)}{12}}} \] (6.1)

is compared with the critical value for a confidence level of 0.05 shown in Altman’s Z-Score table [43]. This table displays the values of the normal distribution parametrized for different confidence levels.

If the absolute value of \( z \) is bigger than the critical value, for a one-tailed test, then the null hypothesis is rejected. If the null hypothesis is rejected, then the mean of the populations, in this case the experimental and control group, are significantly different.

The way of collecting data, explained in section 6.4.1, is known as simple random sampling with replacement. Each user is randomly chosen from a large data set, in this case Twitter’s public timeline. Therefore, each user has the same probability of being chosen, ensuring independent sample values [44–47].

6.4.5 Validity Evaluation

For this study we have considered four validity threats: internal, external, construct and statistical conclusion [48].

Internal validity stands for the extent to which a causal conclusion can be made and how the variables in the experiment are manipulated. In this case, we have not made a cause-effect claim, we study a potential correlation between the dependent and independent variable. We designed the experiment to show that the increase of followers is only influenced by hashtags, by having a large and random Twitter population. Therefore, an appearance of a confounding variable will be canceled out. This experiment is a natural experiment, in contrast with a true experiment. The reason is that we are observing certain individuals, in this case users, and how their actions lead to a change in the number of followers. Since we are merely observers, we are categorizing users into one or other group, but we are not testing subjects based on some predefined actions.

External validity is defined as: "to what populations, settings and variables can this effect be generalized". In terms of this study, it means if the correlation between hashtags and followers could be generalized to all Twitter users, or, on the other hand, only applies to certain users. Since we gather
users from different periods of the day, during a complete week, and with a completely random procedure, we believe that the results indeed generalize for the complete Twitter population.

Construct validity is "the degree to which a test or program measures what it claims". Therefore, we need to ensure that the data that has been collected, and that the computations that have been made are correct. After gathering the data, we analyzed the data in search for some outliers. Several data points were found that seemed like a mistake, e.g. duplicate users or deleted profiles. We cleaned the data after performing tests on all users, and we also performed some tests to check whether the computations (increase of followers, friends, etc.) were done correctly.

Finally, statistical conclusion validity stands for the degree to which conclusions about the relationship between variables match with the correct use of statistics. We used two statistical tests, first, the Kolmogorov-Smirnov test, which concluded that the data was not normally distributed. Then we chose to use a non-parametric test. The main reason is that parametric tests make no assumption of the probability distribution of the variables. We then chose the Mann-Whitney U test, with the main objective of testing whether two samples, in this case increase of followers with and without hashtag usage, are drawn from the same distribution [49].

6.5 Results and Analysis

6.5.1 Statistical characterization

Table 6.2 shows an overview of different statistics obtained from the dataset. Our goal was to portray a better understanding of the users presented in the dataset that were later analyzed in this study. In particular, we present statistics from the followers, friends and tweets. The top 10% users means that the 10% users with the highest value of some variable were analyzed. Popular users are users with a high number of followers. In this case, the ratio of popular users details how many users with highest increase of followers are actually the ones with the highest number of followers. We calculate this by getting the 10 % users with the highest increase of followers and the 10% users with the highest number of followers and see how many users were in common from the population analyzed. In this case, the population analyzed is the users with positive increase of followers. Top 10% users
with high followers and tweets increase, represent the percentage of users
that have both the highest 10% increase of followers and the highest 10%
increase of tweets. Conclusions obtained from these values are explained in
Section 6.5.2.

Table 6.2: Statistical summary of the data

<table>
<thead>
<tr>
<th></th>
<th>Non-Hashtag Group</th>
<th>Hashtag Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>252,957</td>
<td>249,934</td>
</tr>
<tr>
<td>Followers Increase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum</td>
<td>6,248</td>
<td>17,048</td>
</tr>
<tr>
<td>Mean</td>
<td>0.88</td>
<td>2.38</td>
</tr>
<tr>
<td>Users with 0 followers increase (%)</td>
<td>71 %</td>
<td>64 %</td>
</tr>
<tr>
<td>Users with 1 followers increase (%)</td>
<td>12.12 %</td>
<td>12.82 %</td>
</tr>
<tr>
<td>Users between 2-20 followers increase (%)</td>
<td>8.24 %</td>
<td>13.25 %</td>
</tr>
<tr>
<td>Users with more than 20 followers increase (%)</td>
<td>0.84 %</td>
<td>1.89 %</td>
</tr>
<tr>
<td>Top 10 %</td>
<td>36.22</td>
<td>73.86</td>
</tr>
<tr>
<td>Followers Decrease</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>−2.5</td>
<td>−4.2</td>
</tr>
<tr>
<td>% of users</td>
<td>7.8 %</td>
<td>8.05 %</td>
</tr>
<tr>
<td>Total Followers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>1010</td>
<td>1776</td>
</tr>
<tr>
<td>Top 10 %</td>
<td>20,342</td>
<td>36,080</td>
</tr>
<tr>
<td>Popular users ratio</td>
<td>17.19 %</td>
<td>22.06 %</td>
</tr>
<tr>
<td>Tweets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average Increase</td>
<td>10.35</td>
<td>12.18</td>
</tr>
<tr>
<td>% of users with positive increase of tweets</td>
<td>29.12 %</td>
<td>31.12 %</td>
</tr>
<tr>
<td>Friends</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum of the difference of friends</td>
<td>−36,936</td>
<td>−3,403</td>
</tr>
<tr>
<td>Maximum of the difference of friends</td>
<td>5,712</td>
<td>16,815</td>
</tr>
<tr>
<td>Mean of the difference of friends</td>
<td>0.62</td>
<td>1.61</td>
</tr>
</tbody>
</table>

6.5.1.1 Mann-Whitney U test

Since the samples do not follow a normal distribution, the Mann-Whitney
U test is performed to test the existence of a significant difference between
the control and the experimental group. The numerical values obtained for
this test are displayed in Table 6.3.

Table 6.3: Numerical values for the Mann-Whitney U test

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Control Group</th>
<th>Experimental Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group size</td>
<td>$n_1 = 252,957$</td>
<td>$n_2 = 249,934$</td>
</tr>
<tr>
<td>Group rank</td>
<td>$\sum R_1 = 4.23 \cdot 10^{10}$</td>
<td>$\sum R_2 = 8.42 \cdot 10^{10}$</td>
</tr>
<tr>
<td>U Parameter</td>
<td>$U_1 = 5.29 \cdot 10^{10}$</td>
<td>$U_2 = 1.03 \cdot 10^{10}$</td>
</tr>
</tbody>
</table>
We compute $U$ as $U = \min\{U_1, U_2\}$, where the values of $U_1$ and $U_2$ are: $U_1 = 5.29 \cdot 10^{10}$ and $U_2 = 1.03 \cdot 10^{10}$.

Since $U_2 < U_1 \Rightarrow U = U_2 = 1.03 \cdot 10^{10}$. Then, we compute the value of $z$, obtaining: $|z| = 414.27$. Since $z_{0.05} = 1.65 \Rightarrow |z| > z_{0.05}$

The absolute value of $z$ is bigger than $z_{0.05}$, therefore the null hypothesis is rejected and the alternative hypothesis is supported. The mean of the experimental group is significantly higher than the mean of the control group.

### 6.5.2 Data exploration

The goal of this section is to check the relationship between several control variables and the increase of followers, to discover if they are be related. We also portray interesting findings from the users in the dataset.

Popular users are users with a high number of followers. In this study, we characterize these users as the ones with the 10% highest number of followers. From the results shown in Table 6.2, we can see that, on average, users tweeting with hashtags have a higher number of followers, 36,080 against 20,342. This means that there are more popular users tweeting with hashtags that without. The next step is to check if there is a relationship between popular users and the increase of followers. For that reason, we calculate the percentage of users that have both a high increase of followers and a high number of followers. For the non-hashtag case, 17.19% users are the top 10% in both followers increase and number of followers. This number increases by a 5% in the hashtag scenario. Based on this results, we can conclude that even though hashtag users have more popular users, those are increasing their number of followers only 5% more than non-hashtag users, and they only represent a 22% of the users increasing high the number of followers.

In relation to the increase of followers, we can observe how on average, users tweeting with hashtags increase more than double their number of followers that users without hashtags. This matches with the results of our study, that defines a correlation between tweeting with hashtags and the increase of followers. In order to understand how this increase is distributed, we have computed how many users, as a percentage of the total number of users, actually increase zero followers, one, between 2 and 20 and more than 20. What the results show is that a 71% of the users do not increase their
followers in the non hashtag group and a 64% do not increase in the hashtag group. Then this number of users decreases while the number of followers increases, i.e. there are less users increasing a lot their number of followers, but those that tweet with hashtags have always a higher increase than users that tweet without. An interesting phenomenon is that users tweeting with hashtags are also decreasing their followers more, on average, than users tweeting without hashtags. This suggests visibility.

We believe that users tweeting with hashtags are more visible in Twitter. This claim is supported by two facts. First, the fact that they are both loosing and gaining more followers, which suggests that they have more presence in Twitter. Second, the fact that, on average, users tweeting with hashtags are also tweeting more than users tweeting without, 10.35 against 12.18 increase of tweets. Therefore hashtag users are more active than non-hashtag users. So the reason why they are more visible could be because of their activeness and their hashtag use. The question to answer, is then:
Figure 6.3: **Average decrease of followers with and without hashtags.** This figure shows a comparison between the decrease of followers of users tweeting with hashtags and the increase of followers of users tweeting without hashtags. This comparison is per iteration, that is every time the information from each user is updated.

then: **Do users tweeting with hashtags increase more their followers because they are tweeting more (12.18 vs 10.35) rather than their use of hashtags?**

They way we controlled for this possible occurrence is by calculating how many users that have a high increase of followers are actually tweeting at a high rate. Since, if the claim were true, then users tweeting at a higher rate would have a higher increase of followers. We calculate the 10% users with the highest increase of followers, the 10% with the highest increase of tweets and see how many users are in common. The results show that 29.12 % of non-hashtag users have both a high increase of tweets and followers, and that 31.12% of hashtag users have both a high increase of tweets and followers. The main conclusion extracted from these results is that there is no apparent relationship between tweeting at a high rate and having a high increase of followers. If it were, then the difference between hashtag and non-hashtag group would be higher than 2%, because we already know that more users are increasing their followers for the hashtag group (74% vs
Hashtags and followers: An experimental study of the online social network Twitter

6.5.3 Trend analysis

This section illustrates analyses between certain features presented in the data set obtained from Twitter. The main general trend and the key finding in this study is portrayed in Figure 6.1. This figure represents that on average, users that tweet with hashtags have a higher increase of followers than users that tweet without hashtags. This figure matches with the results shown in Table 6.2, where the average increase of followers for users with hashtags is 2.38 and the average increase of followers for users without hashtags is 0.88. Figure 6.2 shows a comparison between the same increase of followers in the hashtag and non-hashtag usage, but separates the increase of followers into different iterations or updates. It can be observed that the highest increase occurs in the first iteration, during the first 12 minutes after the user has published the tweet. Bray [40] stated that a tweet is popular during the first 18 minutes after it has been published. For that reason, it makes sense that the highest increase of followers happens during the first 12 minutes, since after the next iteration, in the minute 24, the tweet’s popularity will have decreased. Also, looking at the blue line that represents the increase of followers for users with hashtags, we can observe that the last update is almost flat. This complements the study by Lardinois [39], where they discovered that the lifespan of a tweet was one hour. Moreover, Figure 6.3 shows how users tweeting with and without hashtag decrease their number of followers. We analyzed the behavior of those users that loose followers and concluded that except from the fourth and fifth update, both types of users decrease their followers in the same way. However, users tweeting with hashtags decrease significantly more their number of followers in the fourth update.

Figure 6.4 shows the relationship between the total number of followers and the total number of friends. The axis are zoomed for a clearer view of the plot. In this figure we can identify two types of Twitter users, already explained in Section 6.3, differentiated between two clusters. The first cluster represents users with high number of followers and low number of friends. We can see this cluster in all the points that move around the X axis but have low values of Y, forming a horizontal line. These users are users that have a lot of followers but that do not follow other users. Users such as celebrities can follow this type of trend, since a lot of fans are following
them but they only follow some friends back. This is called preferential attachment, and was discovered by Barabási et al. [9] in 1999. The second cluster is formed by the points with a linear relationship between followers and friends. The cluster can be observed by looking at the points in the figure that form a diagonal line across the frame. These users could be matched to the follower fallacy [3]. As explained before, users follow other users that follow them just for etiquette or being polite, ending up with similar number of followers and friends. Another interesting characteristic of this figure is the fact that there are seldom users with a high number of friends and a low number of followers. There are some outliers that become visible if we zoom out the figure. Finally, another group of users are the ones with a more random relationship between followers and friends that do not really follow any pattern.

Figure 6.4: **Total number of followers against total number of friends.** Comparison between the total number of followers and friends for users that tweet with and without hashtags. Both users with and without hashtags present a similar trend, in the form of two different clusters. The first cluster is formed by the points that create a horizontal line, and the second cluster by the points that create a linear and diagonal line.

Figures 6.5, 6.6 and 6.7 detail relationships between the increase or
decrease of followers and the number of hashtags that the user tweeted with. As we can see from Figure 6.5, most of the users that tweeted with hashtags tweeted with one hashtag. The number of users decreases as the number of hashtags increases. The goal with this Figure is to portray how many hashtags were used by all users, to then analyze it together with Figures 6.6 and 6.7. Having a lot of users tweeting with a specific number of hashtags does not mean that they increase a lot their followers, but if we check the average increase and decrease of followers per hashtag, it can gives us an overview on how these two measures are related. We can observe that tweeting with one hashtag had, on average, the second highest increase of followers and that users tweeting with 6 hashtags had the highest increase of followers, which is not expected. On average, the higher the number of hashtags the lower the number of followers. In an online study it was suggested that using more than one or two hashtags will drop the user engagement by 17% \(^7\). This claim was also presented in the work by Hutto et al. [27]. If we analyze the decrease of followers from Figure 6.7, tweeting with one hashtag had a higher decrease than the baseline of not tweeting with any hashtag. The higher the usage of hashtags the lower the decrease, although since the change from one hashtag to another is low, the decrease does not seem to be dependent on the number of hashtags. Maybe users tweeting with a lot of hashtags do get less followed but they do not get unfollowed. The final remark is that tweeting with 5 hashtags get a significant decrease of followers. This could be related to some bot actions, were they are tweeting with these amount of hashtags and users unfollow them immediately.

In terms of connectivity, we have analyzed the tweets that are replies or mentions to other users. These tweets represent an 8.15% of the total users that tweet with hashtags. From these 8.15%, the aim was to understand the connection between them. For example, one finding could be that almost all tweets were from the same community, therefore not being generalizable to the complete Twitter population. Figure 6.8 represents those users that tweeted a reply or a mention to another user in their tweets and their connection. The size of the node is proportional to the number of connections to that users. All users with less connections that 4 were removed from the graph to make it more understandable. What can be extracted is that there

\(^7\)https://blog.bufferapp.com/a-scientific-guide-to-hashtags-which-ones-work-when-and-how-many
6.5. Results and Analysis

Figure 6.5: **Number of users with a positive, negative or zero increase of followers against the number of hashtags.** This figure shows the number of users that tweeted with X number of hashtags.

are 2 main groups of users tweeting to a specific user. Those two groups are represented by the blue and the red node. We expect these nodes to be famous users, being that the reason why they receive so many tweets to them. After analyzing them in the database we discovered that the blue node is the Twitter account from the music band *5 Seconds of Summer* (@5SOS) and that the red node corresponds to one of the members of that band (@ashton5sos). Apart from these nodes, there is not an apparent connection between the rest of the users.

A final discovery extracted from the data is that tweets that contain hashtags contain also more URLs than tweets without hashtags. This matches with the claim from Hutto et al. [27], where they found a correlation between URLs and follower growth. Based on this relationship, hashtags and URLs could also be correlated, being this a potential study that we propose as future work.

In summary, the main conclusions extracted from these analyses are the following:
6. Hashtags and followers: An experimental study of the online social network Twitter

Figure 6.6: **Average increase of followers against the number of hashtags.**
The average increase of followers of users tweeting with a certain (x) number of hashtags. On average, the higher the number of hashtags the lower the increase of followers.

- Users that tweet with hashtags have a tendency to increase more their number of followers than users that tweet without hashtags.
- There is a relationship between the number of hashtags and the increase of followers.
- There is a relationship between the number of friends and the number of followers.
- There is a relationship between hashtags and URL usage.

### 6.6 Conclusions

The goal of this study was to determine whether the addition of hashtags to tweets produces new followers. For this reason, we performed a natural experiment in which we gathered random users that tweeted with and without hashtags for a period of 7 days, obtaining a total of 502,891 users.
The next step was to compute the difference in the number of followers in 1 hour slots. Since the data was not normally distributed, we performed a non-parametric test to find out whether the increase of followers was significantly different for users tweeting with and without hashtags. The results show that users tweeting with hashtags have a significant higher increase in the number of followers than users tweeting without hashtags.

Moreover, we extracted several conclusions after performing an analysis on the data. We showed that, on average, users increased more their number of followers during the first 12 minutes after they tweeted. At the same time, users tweeting with hashtags did not increase their number of followers after the minute 48, the 4th update. These numbers indicate the possible lifespan of a tweet, having its peak during the first 12 to 18 minutes, as explained in Section 6.5. Therefore, if companies want to target specific clients, they should be aware that the visibility of their tweets will significantly decrease after the first 12-18 minutes. In addition, by analyzing the connection between the number of followers and friends, we also discovered that apart
Figure 6.8: **Users connectivity** This network represents the connection between users tweets. Each connection represents a tweet that was mentioning or replying to another user. The node’s size is proportional to the number of connections to such node. 

from the standard average user, there are two more types of users. The first type could be a celebrity or a famous person, since they have a lot of followers but they do not follow that many users. The second type are users with a linear relationship between friends and followers, having these users a similar number of friends and followers. Lastly, we also discovered the possible relationship between the number of hashtags and the increase of followers. On average, the increase of followers decreases when the number of hashtags increases. Therefore, if a company wants to efficiently target their clients, the data suggests that they should use two or less hashtags in
their tweets.

We believe that the presented discoveries give a better understanding of users behavior inside Twitter by portraying correlations between certain features. Companies could benefit from these results by building more efficient models to target clients. Thus, they can tweet about campaigns with the knowledge that tweeting with hashtags and the number of hashtags do matter for their impact in Twitter. Finally, several suggestions on how to continue this study are presented in the following and final section.

6.7 Future Work

First of all, we suggest that an interesting work could be made to discover which hashtags attract new followers and which do not. Right now we know that hashtags and increase of followers are correlated, but we do not know precisely the type of hashtags that are responsible for this phenomena. For that reason, one option could be to apply machine learning techniques to group hashtags into different types, and discover if there exists specific type of hashtags that produces an increase of followers. Moreover, another option could be to apply natural language processing techniques in order to morphologically analyze each hashtag.

As a second future investigation, we can use the findings of this study to create a predictor model that is able to predict follower formation depending on the tweets and users characteristics. Lastly, as was mentioned in Section 6.5, we could made an experiment with different users to test for a possible correlation between hashtags and URLs in tweets within the message content.

6.8 References


6. Hashtags and followers: An experimental study of the online social network Twitter

on Information and knowledge management. ACM. 2011, pp. 1031–1040.


6. Hashtags and followers: An experimental study of the online social network Twitter


[42] H. B. Mann and D. R. Whitney. “On a test of whether one of two random variables is stochastically larger than the other”. In: The annals of mathematical statistics 18.1 (1947), pp. 50–60.

6. Hashtags and followers: An experimental study of the online social network Twitter


Energy Efficiency Analysis of the Very Fast Decision Tree Algorithm

Eva Garcia-Martin, 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 CO₂ 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 CO₂. 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 CO₂ [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 CO₂ 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: $3.94 \times 1.8 = 7.092 J = Ws = 1.99 \times 10^{-3} \text{ Wh}$.

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 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 \( \epsilon \) in Equation 7.1.

\[
\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}
\]  

(7.1)

This bound states that with probability \( 1-\delta \), 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, \( \delta \)
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 $\Delta G > \epsilon$. $\Delta G$ stands for the difference in information gain between the best two attributes. Thus, if the number of examples $n$ is small, $\epsilon$ will be high, making it harder to split on an attribute unless $\Delta G$ 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 $\epsilon$ 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: $n_{min}$, $\tau$, $\delta$, 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.

$n_{min}$ parameter is the minimum number of examples that the algorithm must see before calculating $\epsilon$ 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 $n_{min}$ is 200. If the value of $n_{min}$ 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 $n_{min}$. Therefore, we predict that the energy would decrease when increasing $n_{min}$, since we would be decreasing the computational effort and time to analyze each instance.

$\tau$ 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 $\tau$ 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 $\tau$ could increase the energy, due to the fact that with lower $\tau$ we make less number of computations.

$\delta$ parameter represents one minus the probability of choosing the correct attribute to split on. If $\delta$ 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 $\delta$ will not be as high as when removing $\tau$, and the probability of a correct split is lower, our assumption is that the accuracy will be lower when $\delta$ increases. At the same time, if $\delta$ 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 $\delta$ 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 $\epsilon$, 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>
<thead>
<tr>
<th>PARAMETER</th>
<th>MODIFICATION</th>
<th>ACC</th>
<th>ENERGY</th>
<th>NODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{min}$</td>
<td>Increase</td>
<td>Lower</td>
<td>Lower</td>
<td>Fewer</td>
</tr>
<tr>
<td></td>
<td>Decrease</td>
<td>Higher</td>
<td>Higher</td>
<td>More</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Increase</td>
<td>Higher</td>
<td>Higher</td>
<td>More</td>
</tr>
<tr>
<td></td>
<td>Decrease</td>
<td>Lower</td>
<td>Lower</td>
<td>Fewer</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Increase</td>
<td>Lower</td>
<td>†</td>
<td>More</td>
</tr>
<tr>
<td></td>
<td>Decrease</td>
<td>Higher</td>
<td>†</td>
<td>Fewer</td>
</tr>
<tr>
<td>MEM1</td>
<td>100KB</td>
<td>Lower</td>
<td>Lower</td>
<td>Fewer</td>
</tr>
<tr>
<td></td>
<td>2GB</td>
<td>Higher</td>
<td>Higher</td>
<td>More</td>
</tr>
<tr>
<td>MEM2</td>
<td>ON</td>
<td>Lower</td>
<td>Lower</td>
<td>Fewer</td>
</tr>
<tr>
<td>SPLIT CRIT</td>
<td>S2</td>
<td>†</td>
<td>†</td>
<td>†</td>
</tr>
<tr>
<td>RPA</td>
<td>ON</td>
<td>Higher</td>
<td>Lower</td>
<td>Fewer</td>
</tr>
</tbody>
</table>

†=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>
<thead>
<tr>
<th>Dataset Name</th>
<th>Type</th>
<th>Instances</th>
<th>Nominal attributes</th>
<th>Numeric attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Random tree</td>
<td>Synthetic</td>
<td>1,000,000</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2 Hyperplane</td>
<td>Synthetic</td>
<td>1,000,000</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>3 Poker</td>
<td>Real world</td>
<td>1,025,010</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4 Airlines</td>
<td>Real world</td>
<td>539,383</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

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: \( n_{\text{min}} \), \( \tau \), \( \delta \), memory limits, memory management, split criterion, and removing poor attributes. \( n_{\text{min}} \) was varied from the default value, 200, to a maximum value of 1,700 with steps of 500. \( \tau \) 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. \( \delta \) 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>
<thead>
<tr>
<th>IDX</th>
<th>nmin</th>
<th>τ</th>
<th>δ</th>
<th>MEM1</th>
<th>MEM2</th>
<th>S.CRT</th>
<th>RPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>200</td>
<td>0.05</td>
<td>10^{-7}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>B</td>
<td>700</td>
<td>0.05</td>
<td>10^{-7}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>C</td>
<td>1,200</td>
<td>0.05</td>
<td>10^{-7}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>D</td>
<td>1,700</td>
<td>0.05</td>
<td>10^{-7}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>E</td>
<td>200</td>
<td>0.01</td>
<td>10^{-7}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>F</td>
<td>200</td>
<td>0.09</td>
<td>10^{-7}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>G</td>
<td>200</td>
<td>0.13</td>
<td>10^{-7}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>H</td>
<td>200</td>
<td>0.05</td>
<td>10^{-1}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>I</td>
<td>200</td>
<td>0.05</td>
<td>10^{-4}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>J</td>
<td>200</td>
<td>0.05</td>
<td>10^{-10}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>K</td>
<td>200</td>
<td>0.05</td>
<td>10^{-7}</td>
<td>100KB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>L</td>
<td>200</td>
<td>0.05</td>
<td>10^{-7}</td>
<td>2GB</td>
<td>No</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>M</td>
<td>200</td>
<td>0.05</td>
<td>10^{-7}</td>
<td>30MB</td>
<td>Yes</td>
<td>S1</td>
<td>No</td>
</tr>
<tr>
<td>N</td>
<td>200</td>
<td>0.05</td>
<td>10^{-7}</td>
<td>30MB</td>
<td>No</td>
<td>S2</td>
<td>No</td>
</tr>
<tr>
<td>O</td>
<td>200</td>
<td>0.05</td>
<td>10^{-7}</td>
<td>30MB</td>
<td>No</td>
<td>S1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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 tuning, on one of the datasets. In parallel, we will be measuring how much energy is the execution consuming. Each combination of dataset and parameter tuning 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>
<thead>
<tr>
<th>Table 7.4: Design summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quantity</strong></td>
</tr>
<tr>
<td>Datasets</td>
</tr>
<tr>
<td>Measures</td>
</tr>
<tr>
<td>Parameter configuration</td>
</tr>
<tr>
<td>Executions</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>S</th>
<th>T (s)</th>
<th>P (W)</th>
<th>E (J)</th>
<th>A (%)</th>
<th>N</th>
<th>D</th>
<th>S</th>
<th>T (s)</th>
<th>P (W)</th>
<th>E (J)</th>
<th>A (%)</th>
<th>N</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4.44</td>
<td>8.59</td>
<td>38.10</td>
<td>96.91</td>
<td>1,134</td>
<td>8</td>
<td></td>
<td>5.85</td>
<td>7.39</td>
<td>43.19</td>
<td>90.93</td>
<td>655</td>
<td>12</td>
</tr>
<tr>
<td>B</td>
<td>3.90</td>
<td>7.32</td>
<td>28.57</td>
<td>96.31</td>
<td>661</td>
<td>6</td>
<td></td>
<td>5.65</td>
<td>7.12</td>
<td>40.22</td>
<td>90.99</td>
<td>607</td>
<td>11</td>
</tr>
<tr>
<td>C</td>
<td>3.90</td>
<td>7.10</td>
<td>27.71</td>
<td>96.24</td>
<td>570</td>
<td>7</td>
<td></td>
<td>5.66</td>
<td>7.75</td>
<td>43.90</td>
<td>91.22</td>
<td>575</td>
<td>11</td>
</tr>
<tr>
<td>D</td>
<td>3.82</td>
<td>6.99</td>
<td>26.67</td>
<td>95.91</td>
<td>495</td>
<td>7</td>
<td></td>
<td>5.57</td>
<td>7.77</td>
<td>43.31</td>
<td>90.98</td>
<td>515</td>
<td>11</td>
</tr>
<tr>
<td>E</td>
<td>4.46</td>
<td>8.44</td>
<td>37.63</td>
<td>95.57</td>
<td>699</td>
<td>8</td>
<td></td>
<td>5.36</td>
<td>9.15</td>
<td>49.08</td>
<td>90.57</td>
<td>57</td>
<td>7</td>
</tr>
<tr>
<td>F</td>
<td>4.74</td>
<td>7.35</td>
<td>34.84</td>
<td>97.93</td>
<td>1,541</td>
<td>9</td>
<td></td>
<td>7.42</td>
<td>6.89</td>
<td>51.10</td>
<td>90.38</td>
<td>2,071</td>
<td>19</td>
</tr>
<tr>
<td>G</td>
<td>5.09</td>
<td>7.30</td>
<td>39.21</td>
<td>98.27</td>
<td>2,181</td>
<td>12</td>
<td></td>
<td>9.25</td>
<td>6.87</td>
<td>63.55</td>
<td>89.87</td>
<td>3,863</td>
<td>18</td>
</tr>
<tr>
<td>H</td>
<td>5.20</td>
<td>7.35</td>
<td>34.84</td>
<td>97.93</td>
<td>2,074</td>
<td>11</td>
<td></td>
<td>9.51</td>
<td>6.78</td>
<td>64.45</td>
<td>89.75</td>
<td>3,971</td>
<td>19</td>
</tr>
<tr>
<td>I</td>
<td>4.58</td>
<td>7.90</td>
<td>36.22</td>
<td>96.31</td>
<td>1,403</td>
<td>9</td>
<td></td>
<td>6.32</td>
<td>7.01</td>
<td>44.32</td>
<td>90.82</td>
<td>1,129</td>
<td>12</td>
</tr>
<tr>
<td>J</td>
<td>4.29</td>
<td>8.32</td>
<td>36.60</td>
<td>96.91</td>
<td>1,541</td>
<td>9</td>
<td></td>
<td>5.73</td>
<td>7.58</td>
<td>43.48</td>
<td>91.24</td>
<td>471</td>
<td>11</td>
</tr>
<tr>
<td>K</td>
<td>3.98</td>
<td>7.37</td>
<td>29.31</td>
<td>95.43</td>
<td>1,134</td>
<td>8</td>
<td></td>
<td>5.03</td>
<td>7.60</td>
<td>38.20</td>
<td>90.93</td>
<td>655</td>
<td>12</td>
</tr>
<tr>
<td>L</td>
<td>4.40</td>
<td>8.32</td>
<td>36.60</td>
<td>96.91</td>
<td>1,134</td>
<td>8</td>
<td></td>
<td>5.92</td>
<td>7.17</td>
<td>42.48</td>
<td>90.93</td>
<td>655</td>
<td>12</td>
</tr>
<tr>
<td>M</td>
<td>4.39</td>
<td>7.95</td>
<td>34.91</td>
<td>96.91</td>
<td>1,134</td>
<td>8</td>
<td></td>
<td>5.91</td>
<td>7.12</td>
<td>42.11</td>
<td>90.93</td>
<td>655</td>
<td>12</td>
</tr>
<tr>
<td>N</td>
<td>6.12</td>
<td>6.67</td>
<td>40.79</td>
<td>83.11</td>
<td>1,735</td>
<td>16</td>
<td></td>
<td>5.77</td>
<td>7.60</td>
<td>43.90</td>
<td>90.72</td>
<td>619</td>
<td>11</td>
</tr>
<tr>
<td>O</td>
<td>4.34</td>
<td>7.85</td>
<td>34.08</td>
<td>96.91</td>
<td>1,134</td>
<td>8</td>
<td></td>
<td>5.83</td>
<td>7.24</td>
<td>42.23</td>
<td>90.93</td>
<td>655</td>
<td>12</td>
</tr>
</tbody>
</table>


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 $\tau$. When $\tau$ is increased, intuitively the accuracy should decrease, since you are allowing splits on not so good attributes. However, although
7. Energy Efficiency Analysis of the Very Fast Decision Tree Algorithm

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.

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

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

### 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.

<table>
<thead>
<tr>
<th>$n_{min}$</th>
<th>$\tau$</th>
<th>$\delta$</th>
<th>MEM1</th>
<th>MEM2</th>
<th>S2</th>
<th>RPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\uparrow)</td>
<td>(\uparrow)</td>
<td>(\downarrow)</td>
<td>2GB</td>
<td>100KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>ACC</td>
<td>Dec</td>
<td>Inc</td>
<td>Dec</td>
<td>(\downarrow)</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td></td>
<td>ENG</td>
<td>Dec</td>
<td>(\downarrow)</td>
<td>Dec</td>
<td>Dec</td>
<td>Inc</td>
</tr>
<tr>
<td></td>
<td>NDS</td>
<td>Few</td>
<td>More</td>
<td>Few</td>
<td>(\downarrow)</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td>D2</td>
<td>ACC</td>
<td>(\downarrow)</td>
<td>Dec</td>
<td>Inc</td>
<td>Dec</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td></td>
<td>ENG</td>
<td>(\downarrow)</td>
<td>Inc</td>
<td>Dec</td>
<td>Dec</td>
<td>Dec</td>
</tr>
<tr>
<td></td>
<td>NDS</td>
<td>Few</td>
<td>More</td>
<td>Few</td>
<td>(\downarrow)</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td>D3</td>
<td>ACC</td>
<td>(\uparrow)</td>
<td>Inc</td>
<td>Dec</td>
<td>(\downarrow)</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td></td>
<td>ENG</td>
<td>Inc</td>
<td>(\downarrow)</td>
<td>Inc</td>
<td>(\downarrow)</td>
<td>Inc</td>
</tr>
<tr>
<td></td>
<td>NDS</td>
<td>Few</td>
<td>More</td>
<td>Few</td>
<td>(\downarrow)</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td>D4</td>
<td>ACC</td>
<td>(\downarrow)</td>
<td>Inc</td>
<td>Dec</td>
<td>(\downarrow)</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td></td>
<td>ENG</td>
<td>Dec</td>
<td>(\downarrow)</td>
<td>Dec</td>
<td>(\downarrow)</td>
<td>Inc</td>
</tr>
<tr>
<td></td>
<td>NDS</td>
<td>(\downarrow)</td>
<td>Few</td>
<td>(\downarrow)</td>
<td>(\downarrow)</td>
<td>(\downarrow)</td>
</tr>
</tbody>
</table>

\(\uparrow\)=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. \(\downarrow\)=There is no variation.

### 7.6.2.1 Parameter $n_{min}$

Observing the behavior of the $n_{min}$ 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 $n_{min}$ 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 $n_{min}$ increases is because we are computing less times the value of $\Delta G$, therefore saving power. In terms of energy, when $n_{min}$ 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 $n_{min}$ 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 \( n_{min} \) increases, which is reasonable since we are looking into less batches of data, therefore the tree is computed faster.

### 7.6.2.2 Parameter \( \tau \)

When the value of \( \tau \) increases, so does the accuracy for datasets 1, 3, and 4. Dataset 2 experiences a non-stable accuracy value when increasing \( \tau \). 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 \( \tau \) 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 \( \delta \)

When \( \delta \) 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 \( \delta \) 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 \( \delta \) 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 \( \delta \) decreases, the number of nodes decreases also, as we predicted, since there are less splits. Taking a look into the energy aspect, when \( \delta \) increases energy significantly decreases for datasets 1 and 2. Dataset 4 does not vary energy significantly and dataset 3 increases energy when \( \delta \) 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 tuning 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 $n_{min}$ 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 $n_{min}$ 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 $\delta$, 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


7. Energy Efficiency Analysis of the Very Fast Decision Tree Algorithm


7.8. References


[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.
7.8. References


7. Energy Efficiency Analysis of the Very Fast Decision Tree Algorithm


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 \(G\). \(\epsilon\) 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 $1-\delta$.

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

(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

---

2. [https://team.inria.fr/spirals/](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 its 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: $\text{homogeneous()}$, $\text{label()}$, $\text{bestSplit()}$ and $\text{split()}$. The $\text{homogeneous()}$ function returns true if all the instances of the tree can be labeled with a single class. The $\text{label()}$ 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 $\text{bestSplit()}$ function returns the best attribute to split on. This can be achieved in different ways, such as using the information gain function. The $\text{split()}$ 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 $\text{trainOnInstance()}$, 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 $\text{filterInstanceToLeaf()}$, which sorts the instance to the leaf by following the tests at the nodes. Then, the function $\text{learnFromInstance()}$ 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) $\text{newNominalClassObserver()}$ or

---

112
Figure 8.1: Functions structure of the VFDT implementation.

\[
\begin{align*}
\text{trainOnInstance()} \\
\quad \text{filterInstanceToLeaf()} \\
\qquad \text{learnFromInstance()} \\
\qquad \quad \text{newNominalClassObserver()} \\
\qquad \quad \text{newNumericClassObserver()} \\
\qquad \text{attemptToSplit()} \\
\qquad \quad \text{observedClassDistributionIsPrune()} \\
\qquad \quad \text{getBestSplitSuggestion()} \\
\qquad \quad \text{computeHoeffdingBound()} \\
\qquad \quad \text{newLearningNode()} \\
\qquad \text{newSplitNode()} \\
\qquad \text{setChild()}
\end{align*}
\]

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 \texttt{split()} and \texttt{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 \texttt{homogeneous()} and \texttt{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).

<table>
<thead>
<tr>
<th>VFDT Implementation</th>
<th>VFDT algorithm</th>
<th>Decision Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>filterInstanceToLeaf()</td>
<td>Sort ((x,y)) into a leaf (l) ...</td>
<td>label()</td>
</tr>
<tr>
<td>learnFromInstance()</td>
<td>Label (l) and update statistics</td>
<td>label()</td>
</tr>
<tr>
<td>newNominalClassObserver()</td>
<td>Update statistics</td>
<td>label()</td>
</tr>
<tr>
<td>newNumericClassObserver()</td>
<td>Update statistics</td>
<td>label()</td>
</tr>
<tr>
<td>observedClassDistributionIsPrune()</td>
<td>If the examples seen so far ...</td>
<td>homogeneous()</td>
</tr>
<tr>
<td>computeHoeffdingBound()</td>
<td>Comp. (\overline{G}_t(X_i)). Let (X_b) be the attribute...</td>
<td>\texttt{bestSplit()}</td>
</tr>
<tr>
<td>newSplitNode()</td>
<td>Replace (l) by an internal node</td>
<td>\texttt{split()}</td>
</tr>
<tr>
<td>newLearningNode()</td>
<td>Add a new leaf (l_m) ...</td>
<td>\texttt{split()}</td>
</tr>
<tr>
<td>setChild()</td>
<td>Add a new leaf (l_m) ...</td>
<td>\texttt{split()}</td>
</tr>
</tbody>
</table>

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:
8.4. Experimental Design

- 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>
<thead>
<tr>
<th>Dataset Name</th>
<th>Type</th>
<th>Instances</th>
<th>Attributes</th>
<th>Numeric</th>
<th>Nominal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Tree</td>
<td>Synthetic</td>
<td>10,000,000</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Hyperplane</td>
<td>Synthetic</td>
<td>10,000,000</td>
<td>10</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>LED</td>
<td>Synthetic</td>
<td>10,000,000</td>
<td>24</td>
<td>-</td>
<td>24</td>
</tr>
<tr>
<td>Waveform</td>
<td>Synthetic</td>
<td>10,000,000</td>
<td>21</td>
<td>21</td>
<td>-</td>
</tr>
</tbody>
</table>

### 8.4.3 Parameter Tuning

The parameters that have been varied are shown in Table 8.3. The $n_{min}$ 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 $\tau$ parameter represents the tie threshold. Whenever the difference between the two best attributes is calculated, if this difference is smaller than $\tau$, 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 $\delta$ 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.

<table>
<thead>
<tr>
<th>IDX</th>
<th>nmin</th>
<th>( \tau )</th>
<th>( \delta )</th>
<th>Memory</th>
<th>Leaf prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>200</td>
<td>0.05</td>
<td>( 10^{-7} )</td>
<td>30MB</td>
<td>NBA</td>
</tr>
<tr>
<td>B</td>
<td>700</td>
<td>0.05</td>
<td>( 10^{-7} )</td>
<td>30MB</td>
<td>NBA</td>
</tr>
<tr>
<td>C</td>
<td>1,200</td>
<td>0.05</td>
<td>( 10^{-7} )</td>
<td>30MB</td>
<td>NBA</td>
</tr>
<tr>
<td>D</td>
<td>1,700</td>
<td>0.05</td>
<td>( 10^{-7} )</td>
<td>30MB</td>
<td>NBA</td>
</tr>
<tr>
<td>E</td>
<td>200</td>
<td>0.01</td>
<td>( 10^{-7} )</td>
<td>30MB</td>
<td>NBA</td>
</tr>
<tr>
<td>F</td>
<td>200</td>
<td>0.09</td>
<td>( 10^{-7} )</td>
<td>30MB</td>
<td>NBA</td>
</tr>
<tr>
<td>G</td>
<td>200</td>
<td>0.13</td>
<td>( 10^{-7} )</td>
<td>30MB</td>
<td>NBA</td>
</tr>
<tr>
<td>H</td>
<td>200</td>
<td>0.05</td>
<td>( 10^{-1} )</td>
<td>30MB</td>
<td>NBA</td>
</tr>
<tr>
<td>I</td>
<td>200</td>
<td>0.05</td>
<td>( 10^{-4} )</td>
<td>30MB</td>
<td>NBA</td>
</tr>
<tr>
<td>J</td>
<td>200</td>
<td>0.05</td>
<td>( 10^{-10} )</td>
<td>30MB</td>
<td>NBA</td>
</tr>
<tr>
<td>K</td>
<td>200</td>
<td>0.05</td>
<td>( 10^{-7} )</td>
<td>100KB</td>
<td>NBA</td>
</tr>
<tr>
<td>L</td>
<td>200</td>
<td>0.05</td>
<td>( 10^{-7} )</td>
<td>2GB</td>
<td>NBA</td>
</tr>
<tr>
<td>M</td>
<td>200</td>
<td>0.05</td>
<td>( 10^{-7} )</td>
<td>30MB</td>
<td>NB</td>
</tr>
<tr>
<td>N</td>
<td>200</td>
<td>0.05</td>
<td>( 10^{-7} )</td>
<td>30MB</td>
<td>MC</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th></th>
<th>Random tree dataset</th>
<th>Hyperplane dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I ACC #Leaves E1 E2 E3 ET</td>
<td>ACC #Leaves E1 E2 E3 ET</td>
</tr>
<tr>
<td>A</td>
<td>99.82 11,443 96.11 0.01 1.70 97.83</td>
<td>91.40 22,429 230.58 1.10 18.67 250.36</td>
</tr>
<tr>
<td>B</td>
<td>99.78 8,061 98.48 0.02 0.36 98.85</td>
<td>91.38 21,331 235.61 0.86 5.46 241.93</td>
</tr>
<tr>
<td>C</td>
<td>99.73 6,746 101.09 0.02 0.21 101.33</td>
<td>91.47 20,433 238.90 0.80 3.24 242.94</td>
</tr>
<tr>
<td>D</td>
<td>99.69 5,832 97.83 0.01 0.19 98.03</td>
<td>91.73 19,366 256.26 0.65 2.59 259.51</td>
</tr>
<tr>
<td>E</td>
<td>99.50 7,518 99.83 0.02 3.57 103.42</td>
<td>91.82 1,854 231.76 0.00 35.03 266.79</td>
</tr>
<tr>
<td>F</td>
<td>99.87 13,372 86.41 0.01 0.81 87.24</td>
<td>89.33 44,817 199.80 9.01 8.99 217.81</td>
</tr>
<tr>
<td>G</td>
<td>99.88 15,734 80.54 0.14 0.62 81.30</td>
<td>88.90 60,398 197.61 19.61 6.39 232.63</td>
</tr>
<tr>
<td>H 99.90</td>
<td>16,920 76.12 0.01 0.62 76.75</td>
<td>88.90 60,398 197.61 19.61 6.39 232.63</td>
</tr>
<tr>
<td>I</td>
<td>99.87 12,928 88.04 0.03 1.02 89.09</td>
<td>90.18 31,316 202.99 3.08 12.45 218.52</td>
</tr>
<tr>
<td>J</td>
<td>99.78 10,368 102.70 0.02 1.44 104.15</td>
<td>92.04 18,723 258.59 0.02 22.79 281.42</td>
</tr>
<tr>
<td>K</td>
<td>94.94 806 10.89 0.00 0.00 10.89</td>
<td>83.06 524 15.14 0.00 0.00 15.14</td>
</tr>
<tr>
<td>L</td>
<td>99.82 11,444 100.99 0.01 1.21 102.20</td>
<td>92.29 33,549 420.43 0.05 35.38 455.87</td>
</tr>
<tr>
<td>M</td>
<td>99.67 11,443 49.49 0.01 1.22 50.72</td>
<td>87.41 22,442 128.49 1.06 17.47 147.04</td>
</tr>
<tr>
<td>N</td>
<td>99.83 11,443 48.52 0.02 1.27 49.81</td>
<td>91.41 22,442 132.57 1.06 16.74 150.39</td>
</tr>
</tbody>
</table>

| 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 |


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.
8.5. Results and Analysis

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).

<table>
<thead>
<tr>
<th>LED dataset</th>
<th>Waveform dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Dataset</td>
<td>ACC(%)</td>
</tr>
<tr>
<td>Random Tree</td>
<td>99.43</td>
</tr>
<tr>
<td>Hyperplane</td>
<td>90.10</td>
</tr>
<tr>
<td>LED</td>
<td>74.01</td>
</tr>
<tr>
<td>Waveform</td>
<td>84.55</td>
</tr>
</tbody>
</table>

| AVG         | 74.01   | 4,359   | 569.48| 0.01  | 22.93 | 0.00  | 592.42|

E1(J) = Energy in \( label() \), E2(J) = Energy in \( split() \), E3(J) = Energy in \( bestSplit() \), E4(J) = Energy in \( homogeneous() \), ET(J) = E1+2+E3+E4. E4\(\equiv\)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.

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

\( E1 = \) Energy in \( label() \), \( E2 = \) Energy in \( split() \), \( E3 = \) Energy in \( bestSplit() \), \( E4 = \) Energy in \( homogeneous() \), \( ET = E1+E2+E3+E4 \).
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
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. \( \tau \) and \( \delta \) 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 \( \delta \), the lower the confidence, leading to more splits and a reduction in energy. \( \tau \) represents how separate the attributes can be in order to split. The higher the \( \tau \), 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 \( n_{\text{min}} \) 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 \( \delta \) and \( \tau \) 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


8. IDENTIFICATION OF ENERGY HOTSPOTS: A CASE STUDY OF THE VERY FAST DECISION TREE


8.7. References


Hoeffding Trees with $n_{min}$ adaptation

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

Abstract

Machine learning software accounts for a significant amount of energy consumed in data centers. These algorithms are usually optimized towards predictive performance, i.e., accuracy, and scalability. This is the case of data stream mining algorithms. Although these algorithms are adaptive to the incoming data, they have fixed parameters from the beginning of the execution, which lead to energy hotspots. We present dynamic parameter adaptation for data stream mining algorithms to trade-off energy efficiency against accuracy during runtime. To validate this approach, we introduce the $n_{min}$ adaptation method to improve parameter adaptation in Hoeffding trees. This method dynamically adapts the number of instances needed to make a split ($n_{min}$) and thereby reduces the overall energy consumption. We created an experiment to compare the Very Fast Decision Tree algorithm (VFDT, original Hoeffding tree algorithm) with $n_{min}$ adaptation and the standard VFDT. The results show that VFDT with $n_{min}$ adaptation consumes up to 89% less energy than the standard VFDT, trading off a few percent of accuracy. Our approach can be used to trade off energy consumption with predictive and computational performance in the strive towards resource-aware machine learning.

9.1 Introduction

Large-scale data centers account for a significant share of the energy consumption in many countries [1]. The number of data centers and the computational demand is rapidly increasing due to the rate at which data is generated and processed. Although machine learning algorithms are responsible for some part of that computation, since they are introduced in
9. **Hoeffding Trees with nmin adaptation**

Almost all application domains, they are hardly optimized w.r.t. their energy consumption. State-of-the-art algorithms that can have an impact on the energy consumption of data centers are streaming algorithms, since they are designed to run continuously without much interruption.

Although data stream mining algorithms adapt the decision model based on the incoming data, e.g. concept drift adaptation, the parameters of such algorithms are fixed from the beginning of the execution. Having fixed parameters can make an algorithm work inefficiently, thus consuming more computing resources, which is one of the reasons for a higher energy consumption. We present dynamic parameter adaptation for data stream mining algorithms to trade-off energy efficiency against accuracy during runtime.

To demonstrate the general approach, we introduce the nmin adaptation method to improve parameter adaptation in Hoeffding trees. Hoeffding tree algorithms evaluate if nmin instances observed at a node are enough to make a confident split. They calculate the information gain of the attributes based on those nmin instances and see if there is an attribute that yields the highest information gain. However, if those instances are not enough to make a split, the algorithm would have consumed unnecessary energy on making such operations. Since the value of nmin is fixed, we propose nmin adaptation to adapt the value of nmin depending on the incoming data, to ensure that the algorithm calculates the best attributes only when there will be a split. This reduces the amount of computation related to calculating information gain of all the attributes, thus reducing its energy consumption. This method has the following properties: i) adaptive to the characteristic of incoming data, ii) independent for each tree node, and iii) can be applied to any Hoeffding tree algorithm.

We have designed an experiment that compares the Very Fast Decision Tree (VFDT) algorithm [2], i.e. the first Hoeffding tree algorithm, to VFDT with nmin adaptation (VFDT-nmin) on seven datasets with three nmin values. The results show that VFDT-nmin reduces the energy consumption significantly, yielding an average of 23% and up to 89% energy reduction for the studied datasets. The predictive performance, i.e. the accuracy, is only decreased slightly by this energy reduction (1% average loss for VFDT-nmin in comparison to VFDT).

Designing machine learning algorithms from an energy efficiency perspective can have a significant impact on the overall energy consumed worldwide,
from data centers to devices that run on batteries, such as embedded systems or cellphones.

The paper is organized as follows. The background and motivation are presented in Section 9.2. The problem description, together with the solution to this problem, i.e. our contribution, is presented in Section 9.3. Section 9.4 presents the experimental design and the results and analysis. Section 9.5 concludes the paper with the significance and impact of our work.

9.2 Background and Motivation

This section explains the fundamentals of VFDT, by providing the details of how the algorithm works, mapping those details to the pseudocode of Alg. 3. It finishes by explaining different studies in green computing and resource-aware machine learning.

9.2.1 VFDT

Very Fast Decision Tree [2] 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. Each leaf also stores the instances observed so far. After \( n_{min} \) instances are read at that leaf, the algorithm calculates the information gain (\( G \)) from all observed attributes. The difference in information gain between the best and the second best attribute (\( \Delta G \)) is compared with the Hoeffding Bound [3] (\( \epsilon \)). If \( \Delta G > \epsilon \), 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 on that branch. If \( \Delta G < \epsilon < \tau \), a tie occurs, splitting on any of the two top attributes, since they have very similar information gain values. The Hoeffding Bound (\( \epsilon \)),

\[
\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}
\]

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 \( 1 - \delta \).
The pseudocode of VFDT is given in Algorithm 3. The algorithm initializes the tree with a single leaf, the root, and with a stream of instances \( I \) and a set of attributes \( X \). Lines 6-9 are computed for every instance. There, the algorithm: i) reads an instance, ii) sorts it into a leaf by following the decisions at the nodes, and iii) updates the statistics at that leaf. Lines 11-20 are computed every \( n/n\text{min} \) instances. In that block, the algorithm checks the difference between \( X_a \) and \( X_b \) (best attributes). If the difference is larger than \( \epsilon \), there is a split on \( X_a \) with confidence \( 1 - \delta \). On the other hand, if the difference on information gain between \( X_a \) and \( X_b \) is smaller than \( \tau \), there is a split on \( X_a \), since the attributes are equally good. Finally, if those conditions are not met (line 20), then the algorithm calculates the information for each attribute and compares it with the best one. If the difference is higher than \( \epsilon \), that attribute is disabled.

Algorithm 3 VFDT: Very Fast Decision Tree

1: \( HT \): Tree with a single leaf (the root)
2: \( X \): set of attributes
3: \( G(\cdot) \): split evaluation function
4: \( \tau \): hyperparameter set by the user
5: \textbf{while} stream is not empty \textbf{do}
6: \hspace{1em} Read instance \( I_i \)
7: \hspace{1em} Sort \( I_i \) to corresponding leaf \( l \) using \( HT \)
8: \hspace{1em} Update statistics at leaf \( l \)
9: \hspace{1em} Increment \( n_l \): instances seen at \( l \)
10: \hspace{1em} \textbf{if} \( n\text{min} \leq n_l \) \textbf{then}
11: \hspace{2em} Compute \( G_l(X_i) \) for each attribute \( X_i \)
12: \hspace{2em} \( X_a, X_b = \) attributes with the highest \( G_l \)
13: \hspace{2em} \( \Delta G = G_l(X_a) - G_l(X_b) \)
14: \hspace{2em} Compute \( \epsilon \)
15: \hspace{2em} \textbf{if} \( (\Delta G > \epsilon) \) or \( (\epsilon < \tau) \) \textbf{then}
16: \hspace{3em} Replace \( l \) with a node that splits on \( X_a \)
17: \hspace{3em} \textbf{for} each branch of the split \textbf{do}
18: \hspace{4em} New leaf \( l_m \) with initialized statistics
19: \hspace{2em} \textbf{else}
20: \hspace{3em} Disable attr \( \{X_p | (G_l(X_p) - G_l(X_a)) > \epsilon \} \)
21: \hspace{1em} \textbf{end while}

We now discuss the computational complexity of the VFDT. Suppose that \( n \) is the number of instances and \( m \) is the number of attributes. The
algorithm is a loop over \( n \) iterations. Every step between 6 and 9 require execution time that is proportional to \( m \). In the worst case scenario the computational complexity of step 7 is \( O(m) \) according to [2]. 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. Clearly, 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 \( n/n_{\text{min}} \). 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 \( O(n/n_{\text{min}} \cdot m) \). The total computational complexity of the VFDT is \( O(n \cdot m) + O(n/n_{\text{min}} \cdot m) \) and \( n >> n_{\text{min}} \), 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. One of the most common techniques used in all modern processors to reduce their power consumption is DVFS (dynamic voltage frequency scaling). Reams [4] gives a good overview of energy-efficiency in computing for different platforms: servers, desktops and mobile devices. The author also propose 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 computing looks into different aspects apart from computer hardware that can affect the energy consumption of computers [5], from efficient ways to cool data centers, to optimize software implementations in terms of energy efficiency. Since there has not been much interest on developing energy efficient software, a case study has been created to increase the awareness on energy consumption in software [6], showing that energy awareness could be increased by providing the appropriate stimuli.

In relation to big data, data centers, and cloud computing, there have been several studies that design methods for energy-efficient cloud computing [7, 8]. One approach is to use machine learning to predict the workload of data centers and to improve scheduling decisions to reduce the energy
9. Hoeffding Trees with $nmin$ adaptation

consumption [9]. A similar approach is used by Google Deep Mind to reduce the energy used in cooling the data centers [10]. These studies focus 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.

Machine learning researchers have been focusing on improving their algorithms towards scalability, memory and computational efficiency. Data stream mining algorithms analyze data with an aim of reducing the memory usage, by reading the data only once without storing it. Examples of efficient algorithms are the VFDT [2], and a KNN streaming version with self-adjusting memory [11]. In data stream mining, there have been two distinctive directions of research during the past years. One focusing on ensemble models [12], and the other on concept drift adaptation [13, 14].

Regarding machine learning and energy efficiency, there has been a recent surge 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 [15, 16]. One research group in TU Dortmund focuses on resource-aware machine learning [17] from different perspectives, ranging from efficient resource utilization [18], to building algorithms able to run on embedded systems.

Closely related is the work done on building energy-efficient deep neural networks [19]. They develop a model where the energy cost of the principal components of a neural network is defined, used for pruning a neural network without reducing accuracy. While the few papers on resource-aware machine learning are either focused on hardware optimizations or on deep learning, we focus on reducing the energy consumption of Hoeffding tree algorithms, improving the algorithm design, in general for all hardware platforms.

Finally, regarding decision trees in streaming scenarios, most of the extensions to the VFDT have been focused on concept drift adaptation [14], where the number of examples needed to train each node to improve predictive performance without sacrificing memory usage is estimated. While these algorithms have not been designed with energy efficiency in mind, other work has been done on evaluating the energy efficiency of the VFDT [20]. The authors present how energy is consumed on the VFDT and how energy varies based on parameter tuning. The $nmin$ adaptation method, our proposed solution, extends this work by proposing a generic method for Hoeffding trees to adapt the value of $nmin$ to the incoming data, introducing the
9.3 Methods and Technical Solutions

9.3.1 Problem: Fixed parameters

The problem addressed in this paper is how having fixed parameters in data stream mining algorithms can lead to those algorithms being energy inefficient. In particular, we focus on the VFDT algorithm and on the $n_{min}$ parameter, and how having this parameter fixed from the beginning of the execution leads to energy hotspots.

$n_{min}$ is a parameter that decides the number of instances that a leaf needs to observe before computing the operations to check for a split. VFDT, presented in Algorithm 3, calculates whether to split when $n_{min}$ instances have been observed at leaf $l$. However, if $n_{min}$ instances are not enough to make a confident split, (because the difference between the best attributes is smaller than $\epsilon$) the operations to calculate the best attributes have been unnecessarily computed (lines 10-14 in Alg. 3). A fixed value of $n_{min}$ means that VFDT is not flexible to the characteristics of the data, i.e. it assumes that $n_{min}$ is the same over the entire stream.

Calculating the operations related to splitting in an inefficient way directly links to an increase in the energy consumed by the algorithm. To test our assumption that calculating the best attributes is an energy-consuming operation, and how having a fixed $n_{min}$ is directly affecting the overall energy consumption of the VFDT, we create an experiment to discover the functions of the VFDT that consume the most energy. The results of this energy consumption patterns are shown in the following section.

9.3.2 Energy Consumption of the VFDT

We have measured the energy consumed by each function of the VFDT algorithm under different configurations, shown in Table 9.1. All configurations use the random tree synthetic dataset with 10k instances, generated with the random tree generator from Massive Online Analysis (MOA) framework [21]. We measure the energy consumed by the VFDT with different values of $n_{min}$ and different number of numerical and nominal attributes. To evaluate the energy consumption, we use Sniper [22] together with McPAT [23]. Sniper is a parallel x86 simulator, which allows to estimate the energy consumed.
by the algorithm running on the emulated platform. To profile the energy consumed by each function, we inject the code with \texttt{SimMarker()} function calls around each function of interest. McPAT then outputs the energy consumed in each marked region. Table 9.2 explains the profiled functions, mapped to the pseudocode of Alg. 3.

Table 9.1: Configurations to calculate the energy consumption of the VFDT functions, using the Random tree synthetic dataset [21], with a total of 10k instances. The energy consumed on each run is shown in the rightmost column.

<table>
<thead>
<tr>
<th>Config</th>
<th>#Nom Attr</th>
<th>#Num Attr</th>
<th>nmin</th>
<th>Energy (J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>5</td>
<td>200</td>
<td>6.63</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>0</td>
<td>200</td>
<td>1.00</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>5</td>
<td>200</td>
<td>6.14</td>
</tr>
<tr>
<td>D</td>
<td>50</td>
<td>50</td>
<td>200</td>
<td>49.99</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
<td>5</td>
<td>20</td>
<td>18.17</td>
</tr>
</tbody>
</table>

Table 9.2: Explanation of VFDT functions mapped to Alg. 3

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
<th>Line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Instance</td>
<td>Read instance ( l_i ) from the stream.</td>
<td>6</td>
</tr>
<tr>
<td>Best Attributes</td>
<td>Calculate attributes ( X_a ) and ( X_b ) with the highest information gain.</td>
<td>11 – 13</td>
</tr>
<tr>
<td>Update Stats</td>
<td>Update the statistics at leaf ( l ).</td>
<td>8</td>
</tr>
<tr>
<td>Disable poor att</td>
<td>Disable attributes ( { X_p</td>
<td>(G_l ( X_p )) - G_l ( X_a )) &gt; \epsilon } )</td>
</tr>
<tr>
<td>Discrete Split</td>
<td>Split if the best attribute ( X_a ) is nominal.</td>
<td>16 – 18</td>
</tr>
<tr>
<td>Cont. Split</td>
<td>Split if the best attribute ( X_a ) is numeric.</td>
<td>16 – 18</td>
</tr>
</tbody>
</table>

Table 9.3: VFDT function averaged energy consumption from configurations A-E

<table>
<thead>
<tr>
<th>Rank</th>
<th>Per function call</th>
<th>E (J)</th>
<th>Total Function</th>
<th>E (J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cont. Split</td>
<td>0.25</td>
<td>Read Example</td>
<td>7.12</td>
</tr>
<tr>
<td>2</td>
<td>Discrete split</td>
<td>0.10</td>
<td>\textbf{Best Attributes}</td>
<td>3.50</td>
</tr>
<tr>
<td>3</td>
<td>\textbf{Best Attributes}</td>
<td>0.04</td>
<td>Update Stats</td>
<td>3.46</td>
</tr>
<tr>
<td>4</td>
<td>Disable poor att</td>
<td>0.02</td>
<td>Disable poor att</td>
<td>1.96</td>
</tr>
<tr>
<td>5</td>
<td>Read Example</td>
<td>7.0 ( \cdot 10^{-4} )</td>
<td>Discrete Split</td>
<td>0.72</td>
</tr>
<tr>
<td>6</td>
<td>Update Stats</td>
<td>2.5 ( \cdot 10^{-4} )</td>
<td>Cont. Split</td>
<td>0.34</td>
</tr>
</tbody>
</table>

The results of the energy consumed by the VFDT are shown in Tables 9.1 and 9.3. Table 9.3 shows the most energy consuming functions of VFDT, obtained by averaging the energy consumption of all configurations.
The functions are ranked based on two measurements: the average energy consumed per function call, and the energy consumed in total (average energy/per function call × number of function calls). The results demonstrate that calculating the best attributes is one of the most energy consuming functions. This function calculates the information gain for each attribute, selecting the two attributes, $X_a$ and $X_b$, with the highest information gain. On average, this function is ranked 3 based on the energy per function call, and 2 for the total energy. It is ranked 1 for configuration E, where $n_{\text{min}} = 20$, since it is called every 20 instances, consuming a lot of energy. For the other configurations it is ranked 3 or 4, based also on the total energy. The number of times this function is called is governed by the parameter $n_{\text{min}}$, the parameter that we want to adapt in this study for all Hoeffding tree algorithms.

Table 9.1 shows that setting $n_{\text{min}} = 20$ (configuration E) consumes three times more energy than setting $n_{\text{min}}$ to the default value (configuration A). Continuous splits are twice as expensive regarding energy consumption as discrete splits. Finally, training the model with only numeric attributes (configuration C) is six times more energy consuming than training it with only nominal attributes (configuration B). That is because updating numerical attributes is 20x more energy consuming than updating nominal attributes.

### 9.3.3 Solution: Dynamic parameter adaptation

This section presents the $n_{\text{min}}$ adaptation method for Hoeffding tree algorithms to illustrate dynamic parameter adaptation. In particular for this study, we show how it can be used in the VFDT algorithm. This method adapts the value of $n_{\text{min}}$ depending on the input data to set it to the minimum number of examples that $n_l$ needs to observe to make a confident split. $n_{\text{min}}$ adaptation exhibits two key properties: node independence and data adaptiveness. Each leaf can be considered independent, thus a customized value of $n_{\text{min}}$ can be set up for a leaf depending on the instances observed. This method is sensitive to changes on input data, since it adapts the value of $n_{\text{min}}$ depending on the information gain of the attributes and instances observed so far. Such an adaptive algorithm better reflects the current characteristics of the data by being able to examine the observed information and adjust to the evolution of the data, i.e., it could be considered suitable for scenarios with concept drift.
**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 high value decreases accuracy, and setting nmin to a lower value increases the accuracy at the expense of energy, as it has to calculate the $\bar{G}$ of all attributes even in cases when the tree does not split. We have identified two scenarios that are responsible for not splitting. We set nmin to a different value to address these scenarios and ensure a split. The equations for nmin show how nmin will be different depending on the observed data and the information gain from the observed attributes, thus the adaptation property.

**Scenario 1.** $\Delta \bar{G} < \epsilon$ and $\Delta \bar{G} > \tau$: Figure 9.1, left plot. The attributes are not too similar, since $\Delta \bar{G} > \tau$, but their difference is not big enough to make a split, since $\Delta \bar{G} < \epsilon$. The solution is to wait for more examples until $\epsilon$ (green dot) decreases and is smaller than $\Delta \bar{G}$ (black dot). Following this reasoning, $nmin = \lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot (\Delta \bar{G})^2} \rceil$, obtained by setting $\epsilon = \Delta \bar{G}$ in the Hoeffding bound equation, to guarantee that $\Delta \bar{G} \geq \epsilon$ will be satisfied in the next iteration, creating a split.

**Scenario 2.** $\Delta \bar{G} < \epsilon$ and $\Delta \bar{G} < \tau$ but $\epsilon > \tau$: the top attributes are very similar in terms of information gain, but $\epsilon$ is still higher than $\tau$, as can be seen in Figure 9.1, right plot. The algorithm needs more instances so that $\epsilon$ (green dot) decreases and is smaller than $\tau$ (red dot). Following this reasoning, $nmin = \lceil \frac{R^2 \cdot ln(1/\delta)}{2 \cdot \tau^2} \rceil$, by setting $\epsilon = \tau$. In the next iteration $\epsilon \leq \tau$ will be satisfied, forcing a split.

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 with nmin adaptation is $O(n \cdot m)$.

### 9.3.4 A preliminary study of nmin variation

We have studied the variation of nmin for 3 different datasets: RTreeC, poker, and airline; shown in Figure 9.2. RTreeC is an artificial dataset generated with the MOA random tree generator. It contains 1 million instances, 5 nominal attributes and 5 numeric attributes. We have created this dataset to show how nmin varies in a more realistic dataset with a large
9.4. Empirical Evaluation

Figure 9.1: Variation of $\epsilon$ (Hoeffding bound) with the number of instances. $n_{min}$ adaptation method for scenarios 1 and 2.

number of instances. Poker and airline are real world datasets, detailed in Table 9.4.

Figure 9.2 shows the $n_{min}$ variation for the cases when $n_{min}$ is initially set to 20, 200, 2000. RTreeC and airline show many adaptations of $n_{min}$ when $n_{min}$ is initially set to 20. Each leaf is initially set to $n_{min} = 20$, which is a very small value, thus many leaves will adapt $n_{min}$ to a higher value. There are many leaves adapting $n_{min}$ to 2,763 when Scenario 2 occurs, due to the user defined value of $\tau$ obtained from $n_{min} = \lceil \frac{R^2 \ln(1/\delta)}{2 \cdot \tau^2} \rceil$.

The poker dataset exhibits a different behavior, where $n_{min}$ adapts to a higher value, 30,491. This occurs in Scenario 2, but since the poker dataset has 10 classes, the range $R$ of the Hoeffding bound equation is higher. Finally, looking at the cases where $n_{min} = 2000$ (red), we observe how there is almost no adaptation. VFDT-$n_{min}$ either splits after 2000 instances, or it adapts $n_{min} = 2,763$ or $n_{min} = 30,491$, because the attributes are very similar.

9.4 Empirical Evaluation

9.4.1 Experiment

We have designed an experiment that compares VFDT, and VFDT-$n_{min}$ under different setup scenarios. Namely, we have conducted an experiment with 7 different datasets and 3 different algorithm modifications. The
Figure 9.2: Variation of nmin for nmin initially set to 20, 200, 2000 on datasets: RTreeC, poker, and airline. RTreeC: Random tree synthetic dataset, 1 million instances, 5 nominal and 5 numeric attributes. Poker and airline are explained in Table 9.4. 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.

experiment has been run using Sniper [22], on a Linux machine with an Intel® Xeon® processor (E5-2420 v2), with 32GB of RAM. The implementation of VFDT with nmin adaptation together with the scripts to conduct the experiments are available in the following link: https://www.dropbox.com/sh/3ed421rvqfo5kov/AAAXgdxTwGobDUrwLzEoJhBta?dl=0\textsuperscript{1}.

\textsuperscript{1}A permanent GitHub link will be given once the paper has been reviewed, since at the moment it is a private repository.
9.4. Empirical Evaluation

9.4.1.1 Datasets

We used real and artificial datasets to compare VFDT against VFDT with \textit{nmin adaptation}. Since VFDT-\textit{nmin} is sensitive to changes in the data, we want to test this property by also using datasets with concept drift, inspired from a recent paper with a similar setup [11], where they test their KNN classifier on concept drift scenarios. The datasets are described in Table 9.4.

Table 9.4: Datasets used in the experiment to compare VFDT against VFDT with \textit{nmin adaptation}. We use different sets of data for training and testing. \#Nom: Number of nominal attributes, \#Num: Number of numerical attributes.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Train</th>
<th>#Test</th>
<th>#Nom</th>
<th>#Num</th>
<th>Class</th>
<th>Type</th>
</tr>
</thead>
</table>
| RTreeA         | 67,000  | 33,000 | 5     | 5     | 2     | Artf.
| RTreeB         | 67,000  | 33,000 | 50    | 50    | 2     | Artf.
| Hyperplane     | 67,000  | 33,000 | 0     | 10    | 2     | Artf.
| Waveform       | 67,000  | 33,000 | 0     | 21    | 3     | Artf.
| Poker Hand     | 555,564 | 273,637| 5     | 5     | 10    | Real |
| Electricity    | 30,359  | 14,953 | 1     | 6     | 2     | Real |
| Airline        | 361,387 | 177,996| 4     | 3     | 2     | Real |

The artificial datasets are generated with Massive Online Analysis (MOA) [21]. The RTree datasets are obtained with the Random tree generator function. It is inspired from the dataset proposed by the authors of the original VFDT [2]. 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.

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. More details are given in [13], where the authors developed this dataset to test concept drift scenarios. 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 tested three real datasets, all available from the MOA official website [24]. 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 [25], 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.

Finally, the airline dataset is provided by Elena Ikonomovska [26] and the task is to predict if a given flight will be delayed based on attributes such as airport of origin and airline.

### 9.4.1.2 Algorithms and setups

We compare VFDT and VFDT-\(n\text{\textsubscript{min}}\) for three different variations of \(n\text{\textsubscript{min}}\), shown in Table 9.5. The goal is to compare their energy consumption and accuracy when \(n\text{\textsubscript{min}}\) varies between 20, 200 and 2000. Our assumption is that VFDT-\(n\text{\textsubscript{min}}\) will consume significantly less energy than VFDT, and that will be especially visible when \(n\text{\textsubscript{min}} = 20\). The reason is that VFDT is checking for splits every 20 instances, while VFDT-\(n\text{\textsubscript{min}}\) will increase \(n\text{\textsubscript{min}}\) to a more efficient value after the first 20 instances have been observed, thus reducing the number of times the algorithm checks for splits.

Regarding the evaluation measures, we check the energy consumption and accuracy, to understand if there is a trade-off between both measures. 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 9.4. We have not performed yet prequential evaluation as with this method, however that is planned for future works.

**Table 9.5: Algorithms and parameter settings used in the experiment design.**

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Algorithm</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>VFDT-(n\text{\textsubscript{20}})</td>
<td>VFDT with (n\text{\textsubscript{min}}) adaptation</td>
<td>(n\text{\textsubscript{min}} = 20)</td>
</tr>
<tr>
<td>VFDT-(n\text{\textsubscript{200}})</td>
<td>VFDT with (n\text{\textsubscript{min}}) adaptation</td>
<td>(n\text{\textsubscript{min}} = 200)</td>
</tr>
<tr>
<td>VFDT-(n\text{\textsubscript{2000}})</td>
<td>VFDT with (n\text{\textsubscript{min}}) adaptation</td>
<td>(n\text{\textsubscript{min}} = 2000)</td>
</tr>
<tr>
<td>VFDT(20)</td>
<td>Original VFDT</td>
<td>(n\text{\textsubscript{min}} = 20)</td>
</tr>
<tr>
<td>VFDT(200)</td>
<td>Original VFDT</td>
<td>(n\text{\textsubscript{min}} = 200)</td>
</tr>
<tr>
<td>VFDT(2000)</td>
<td>Original VFDT</td>
<td>(n\text{\textsubscript{min}} = 2000)</td>
</tr>
</tbody>
</table>

### 9.4.2 Results and Discussion

VFDT-\(n\text{\textsubscript{min}}\) and VFDT have been executed with three different \(n\text{\textsubscript{min}}\) values (20, 200 and, 2000) on 7 different datasets, with a total of 42 executions. Each execution is evaluated with respect to energy consumption and classification accuracy. The results are presented in Tables 9.6 and 9.7. Columns \(\Delta_{20}\), \(\Delta_{200}\), and \(\Delta_{2000}\) show the percentage of reduced energy (or accuracy, respectively),
9.4. Empirical Evaluation

Table 9.6: Energy Results measured in joules (J). The columns \( \Delta_i \) represent the percentage of energy reduced by \( \text{VFDT-n}_i \) compared to \( \text{VFDT}_i \), \( i \) being the value of \( n_{min} \). A positive \( \Delta_i \) shows a reduction of energy, a negative \( \Delta_i \) shows an increase of energy. Results in bold represent those cases where \( \text{VFDT-n}_n \) consumes less energy than \( \text{VFDT} \).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>VFDT (J)</th>
<th>20</th>
<th>200</th>
<th>2000</th>
<th>VFDT-n(_{nmin}) (J)</th>
<th>20</th>
<th>200</th>
<th>2000</th>
<th>( \Delta ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTreeA</td>
<td>45.25</td>
<td>21.57</td>
<td>21.65</td>
<td></td>
<td>20.15</td>
<td>19.86</td>
<td>22.52</td>
<td></td>
<td>55.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-4.02</td>
</tr>
<tr>
<td>RTreeB</td>
<td>304.25</td>
<td>182.85</td>
<td>208.80</td>
<td></td>
<td>183.48</td>
<td>174.39</td>
<td>209.83</td>
<td></td>
<td>39.69</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.49</td>
</tr>
<tr>
<td>hyperplane</td>
<td>304.00</td>
<td>67.08</td>
<td>38.03</td>
<td></td>
<td>40.00</td>
<td>38.57</td>
<td>38.29</td>
<td></td>
<td>86.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>42.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.68</td>
</tr>
<tr>
<td>waveform</td>
<td>690.04</td>
<td>41.79</td>
<td>81.79</td>
<td></td>
<td>75.47</td>
<td>75.88</td>
<td>71.68</td>
<td></td>
<td>89.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>46.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12.36</td>
</tr>
<tr>
<td>poker</td>
<td>181.63</td>
<td>50.94</td>
<td>151.25</td>
<td></td>
<td>168.37</td>
<td>163.98</td>
<td>167.94</td>
<td></td>
<td>7.30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-8.64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11.03</td>
</tr>
<tr>
<td>elec</td>
<td>28.90</td>
<td>10.56</td>
<td>8.69</td>
<td></td>
<td>8.32</td>
<td>8.26</td>
<td>8.45</td>
<td></td>
<td>71.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>21.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.76</td>
</tr>
<tr>
<td>airline</td>
<td>93.77</td>
<td>81.80</td>
<td>82.82</td>
<td></td>
<td>77.95</td>
<td>77.95</td>
<td>79.93</td>
<td></td>
<td>16.87</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.49</td>
</tr>
<tr>
<td>Average</td>
<td>235.41</td>
<td>93.80</td>
<td>84.72</td>
<td></td>
<td>81.96</td>
<td>79.84</td>
<td>85.52</td>
<td></td>
<td>52.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.34</td>
</tr>
</tbody>
</table>

Table 9.7: Classification Accuracy Results. The columns \( \Delta_i \) represent the difference in accuracy between \( \text{VFDT-n}_i \) and \( \text{VFDT}_i \). A positive \( \Delta_i \) means that \( \text{VFDT-n}_n \) has a higher accuracy than \( \text{VFDT} \) and vice versa. Results in bold represent those executions where \( \text{VFDT-n}_n \) has a higher accuracy than \( \text{VFDT} \).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>VFDT (%)</th>
<th>20</th>
<th>200</th>
<th>2000</th>
<th>VFDT-n(_{nmin}) (%)</th>
<th>20</th>
<th>200</th>
<th>2000</th>
<th>( \Delta ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTreeA</td>
<td>83.87</td>
<td>83.74</td>
<td>82.75</td>
<td></td>
<td>83.77</td>
<td>83.75</td>
<td>82.75</td>
<td></td>
<td>-0.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.00</td>
</tr>
<tr>
<td>RTreeB</td>
<td>88.40</td>
<td>88.39</td>
<td>85.18</td>
<td></td>
<td>87.81</td>
<td>88.39</td>
<td>85.18</td>
<td></td>
<td>-0.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.00</td>
</tr>
<tr>
<td>hyperplane</td>
<td>76.79</td>
<td>76.98</td>
<td>73.35</td>
<td></td>
<td>73.85</td>
<td>73.89</td>
<td>72.75</td>
<td></td>
<td>-2.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-3.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.60</td>
</tr>
<tr>
<td>waveform</td>
<td>72.55</td>
<td>72.52</td>
<td>73.57</td>
<td></td>
<td>72.47</td>
<td>72.77</td>
<td>72.08</td>
<td></td>
<td>-0.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1.49</td>
</tr>
<tr>
<td>poker</td>
<td>62.52</td>
<td>72.27</td>
<td>67.80</td>
<td></td>
<td>69.93</td>
<td>69.48</td>
<td>67.32</td>
<td></td>
<td>7.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-2.80</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.48</td>
</tr>
<tr>
<td>elec</td>
<td>73.21</td>
<td>75.41</td>
<td>74.73</td>
<td></td>
<td>76.23</td>
<td>76.23</td>
<td>75.36</td>
<td></td>
<td>3.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.62</td>
</tr>
<tr>
<td>airline</td>
<td>57.62</td>
<td>58.28</td>
<td>57.47</td>
<td></td>
<td>57.14</td>
<td>57.45</td>
<td>57.45</td>
<td></td>
<td>-0.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.03</td>
</tr>
<tr>
<td>Average</td>
<td>73.57</td>
<td>75.37</td>
<td>73.55</td>
<td></td>
<td>74.45</td>
<td>74.57</td>
<td>73.27</td>
<td></td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.81</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.28</td>
</tr>
</tbody>
</table>

for corresponding values of \( n_{min} \) (i.e., 20, 200 and 2000), between \( \text{VFDT-n}_n \) and \( \text{VFDT} \).

The first observed trend is that the lower the value of \( n_{min} \), the higher the energy reduction of \( \text{VFDT-n}_n \) in comparison to \( \text{VFDT} \). \( \text{VFDT-n}_n \) adapts the low value of \( n_{min} \) to a larger one, while \( \text{VFDT} \) is evaluating the best attributes more frequently, e.g., every 20 instances. That is the reason why \( \text{VFDT} \) has a higher energy consumption than \( \text{VFDT-n}_n \). Figure 9.3 shows this phenomena by presenting the energy reduction of \( \text{VFDT-n}_n \)
for the artificial and real world datasets. The aforementioned trend is clearly illustrated in the figure. In general, if VFDT is set to a lower \( nmin \) value than the optimal one, VFDT will consume significantly more energy. In the same scenario, VFDT-\( nmin \) would adapt \( nmin \) to a better value, avoiding that overhead of energy consumption.

The second trend is that the energy consumption of VFDT-\( nmin \) is almost independent of the value of \( nmin \). This is shown in Table 9.6. Considering any specific dataset, we can witness that the energy consumption is almost the same for when \( nmin \) is initially set to 20, 200, 2000. This behavior is logical, since VFDT-\( nmin \) adapts \( nmin \) depending on the data, independently of the initial \( nmin \).

The last trend is that accuracy is almost identical between VFDT-\( nmin \)
9.5. Significance and Impact

and VFDT. While VFDT-\textit{nmin} reduces the energy consumption significantly, accuracy is not affected. This is shown in Figure 9.4, where we observe the similar accuracy scores between VFDT-\textit{nmin} and VFDT. VFDT-\textit{nmin} obtains even a higher accuracy than VFDT in datasets \textit{poker} and \textit{elec} when \textit{nmin} = 20, while still having a lower energy consumption than VFDT (seen in Table 9.7).

Generally, VFDT-\textit{nmin} consumes 23.25\% less energy than VFDT, sacrificing less than 1\% of accuracy. The highest energy reduction occurs when \textit{nmin} = 20 on the waveform dataset, which is an 89\% reduction. On average, VFDT-\textit{nmin} obtains 0.06\% less accuracy than VFDT, with 3\% maximum of accuracy loss in the hyperplane dataset, and a 7.4\% maximum of accuracy gain in the poker dataset. Concept drift datasets have achieved higher energy reductions than the other datasets (see Figure 9.3, red and turquoise dots). This is due to the fact that both concept drift datasets present attributes that are quite similar between each other, thus forcing \textit{nmin} to adapt to the number of instances needed for $\epsilon = \tau$ to be satisfied.

Finally, we observe that VFDT-\textit{nmin} consumes more energy than VFDT for the poker dataset when \textit{nmin} = 200. The reason is that \textit{nmin} adapts to a very high value (30,491 instances, observed in Figure 9.2), and since many attributes are numerical, updating the statistics at each node for that many numerical instances requires a lot of energy. That was also observed in Section 9.3.2, where updating the statistics for numerical attributes consumed 20× more energy than for nominal attributes. An interesting future work is to evaluate the trade-off between the energy consumption of updating the statistics and calculating the best attributes. If \textit{nmin} is set to a very high value, more energy is consumed in updating the statistics, less in calculating the best attributes, but the overall energy consumption might be higher, like in this case. The opposite occurs if \textit{nmin} is set to a low value.

9.5 Significance and Impact

In this paper we introduced dynamic parameter adaptation for data stream mining algorithms to reduce their energy consumption. We have shown how an algorithm with fixed parameters consumes more energy than an algorithm that dynamically adapts its parameters, while maintaining similar levels of accuracy. In particular, we have presented the \textit{nmin adaptation} method for Hoeffding trees. The results have shown that the VDFT with
**Hoeffding Trees with nmin adaptation** can consume up to 89% less energy, affecting accuracy at most by a 3%, in comparison with the standard VFDT.

We believe that this study presents a significant contribution to the field of data stream mining. We illustrate a method that can change the way we currently design this class of algorithms, with a new focus on energy efficiency and dynamic parameter adaptation. Designing state-of-the-art online algorithms using a dynamic parameter adaptation approach can both make algorithms energy efficient and more accurate, since they are more adaptive to the type of data.

Designing energy efficient algorithms can have a great impact globally. First, by reducing the energy consumption of data centers, which is directly linked with creating sustainable and environmentally friendly solutions. Finally, by being able to move machine learning algorithms to embedded systems and battery-powered devices, which is currently a big constraint.

For future work, we aim to evaluate further the **nmin adaptation** method on other Hoeffding tree algorithms, such as the Hoeffding Adaptive Tree ([14]), and on ensembles of Hoeffding trees.

### 9.6 References


