Eric Kenzo Taniguchi Onuki

Bringing Awareness to Energy Consumption in Data Stream Mining

Thesis project presented to the Graduate Program in Informatics of Pontifícia Universidade Católica do Paraná (PUCPR) as partial requirement for the degree of Master in Informatics.

Curitiba
2022

Eric Kenzo Taniguchi Onuki

Bringing Awareness to Energy Consumption in Data Stream Mining

Dissertation project presented to the Graduate Program in Informatics of Pontifícia Universidade Católica do Paraná (PUCPR) as partial requirement for the degree of Master in Informatics.

Major Field: Computer Science

Adviser: Jean Paul Barddal
Co-adviser: Andreia Malucelli

Curitiba
2022

Contents

List of Figures

List of Tables

List of Abbreviations

ICT	Information and Communications Technology
MOA	Massive Online Analysis
TIC	Tecnologia da Informação e Comunicação
AI	Artificial Intelligence
CPU	Central Processing Unit
DDM	Drift Detection Method
EDDM	Early Drift Detection Method
ADWIN	Adaptive Windowing
K-D	K-Dimensional
kNN	k-Nearest Neighbors
VFDT	Very Fast Decision Tree
PHT	Page-Hinkley Test
WEKA	Waikato Environment for Knowledge Analysis
DRAM	Dynamic Random Access Memory
API	Application Programmable Interface
RAPL	Running Average Power Limit

Abstract

This work focuses on bringing awareness to green computing and energy efficiency throughout a data stream mining model's life cycle. Seldom during the model selection is energy efficiency a deciding factor. It usually becomes a relevant metric much later when the model has grown to a point when it consumed too many resources. Energy consumption measuring is often done physically, which may require specific tools and hardware, thus becoming prohibitive and costly. The primary objective is to bring awareness to the energy consumption of data stream models by developing a software method that measures its energy consumption in its development phase. In order to accomplish the objective, it is essential to develop a well-accessible form of measuring energy consumption available for general use that does not require custom build hardware. Therefore, this work comprises (i) an analysis of available hardware and software tools to measure the energy consumption of a data stream mining model, (ii) the definition of a plugin for the Massive Online Analysis (MOA) that measures the energy consumption of a computer via software, and (iii) an evaluation of the energy impact of changing the validation protocol and having a concept drift. Therefore, different classifiers and evaluators were selected and evaluated under stable and concept drifting scenarios to assess energy consumption. The results obtained indicate that data stream classifiers' energy modeling is a complex task. The energy measurement is a straightforward process, but generalizing the results in a mathematical equation seems unfeasible, and the evidence is noticeable in the results. Finally, as a contribution to the community, the energy measurement tool created will be available as a plugin for the MOA software.

Keywords: Green computing; Data stream mining; Energy Consumption Measurement.

Resumo

Este trabalho se concentra em conscientizar a computação verde (*green computing*) e a eficiência energética em todo o ciclo de vida do modelo de mineração de fluxo de dados. Raramente, durante a seleção do modelo, a eficiência energética é um fator decisivo. Geralmente se torna uma métrica relevante mais tarde, quando o modelo cresceu a um ponto em que já consumiu muitos recursos. A medição do consumo de energia geralmente é feita fisicamente, o que pode exigir ferramentas e hardwares específicos, tornando-se proibitivo e caro. O objetivo principal é conscientizar sobre o consumo de energia dos modelos de fluxo de dados, desenvolvendo um método de software que mede seu consumo de energia em sua fase de desenvolvimento. Para atingir o objetivo, é essencial desenvolver uma forma acessível de medir o consumo de energia disponível para uso geral que não exija hardware. Portanto, este trabalho compreende (i) a análise de ferramentas de hardware e software disponíveis para medir o consumo de energia de um modelo, (ii) a definição de um plugin para o Massive Online Analysis (MOA) que mede o consumo de energia de um computador via software, e (iii) a avaliação do impacto energético de alterar o protocolo de validação e ter uma mudança de conceito. Para tanto, diferentes classificadores e avaliadores foram selecionados e avaliados em cenários estáveis e de mudança de conceito para avaliar o consumo de energia. Os resultados obtidos indicam que a modelagem energética dos classificadores de fluxo de dados é uma tarefa complexa. A medição de energia é um processo direto, mas generalizar os resultados em uma equação matemática é inviável e a evidência é perceptível nos resultados. Por fim, como contribuição à comunidade, a ferramenta de medição de energia criada é disponibilizada como plugin para o MOA.

Palavras-chave: Computação Verde; Mineração de Fluxo de Dados; Medição de consumo energético.

Chapter 1

Introduction

Data generation and consumption are gaining momentum in the late years as society is becoming more and more knowledge intensive (BIFET; KIRKBY, 2009). Consequently, each passing second generates and stores more data than ever in human history (MISHRA; YAZICI; MISHRA, 2012). Every sensor available today is potentially recording a particular aspect of the world. Anyone with access to the internet can, with a few clicks and keystrokes, explore a vast world of data that is made available (and at the same time, generate more data through clicks and behaviors).

This massive increase in data generation solved a long time problem machine learning faced: lack of data. In the early days of machine learning, the limiting factor was the insufficient data to produce useful models (HALEVY; NORVIG; PEREIRA, 2009; SUN et al., 2017). Due to this lack of data, models processed the same sample many times, trying to extract the maximum amount of information, at the expense of efficiency.

Traditional machine learning works with data in batches, mainly using databases as the primary source of information (Moulet; Kodratoff, 1995). The massive amount of data generated by today's devices is a huge problem for traditional models. Due to the sheer volume of information available, batch machine learning has many problems to face (RAO et al., 2017). For example, going back to the first example of a user clicking and generating data, the user's actions are time-bound, meaning they mean something at that moment, but they may not mean anything too much time later (HARIRI; MOBASHER; BURKE, 2015), as most batch machine learning algorithms do not take time into account. Another problem is dealing with massive amounts of data. It is impossible to store everything, hoping to process all data every time a model needs to update.

One possible solution is using data streams as the source. However, just changing the data source is not enough. When changing the data source, most strategies need to adapt, as traditional premises do not hold in a data stream scenario. Due to data streams'

nature to be possibly infinite, change over time, and have different degrees of availability (GAMA, 2010), models dealing with them do have many adjustments over their batch counterparts.

This area of study, data stream mining, studies how to generate models utilizing a stream of data as its input. Data stream mining gained much interest in the past decade or so due to increased computational power and data availability. Different from batch machine learning, it does not requires data available at the start of the training. Furthermore, it does require re-training once new data becomes available. When the amount of data tends to be never-ending, possibly infinite, it poses a considerable problem, as storage solutions will eventually run out of space and resources, so most data stream mining models do not save all the data received, having some keeping track of a window of samples, while others do not store any of the input samples.

While data stream mining does solve the problem of ever-growing learning time as the dataset increases, it is still not nearly as efficient as a time-constant learner and predictor (VASEEKARAN, 2017). Therefore, one of the problems faced by data stream mining is dealing with the energy efficiency of the models (KARAX; MALUCELLI; BARDDAL, 2019). Even though data stream mining has a small resource footprint up to a certain point, overall, data stream mining's resource footprint is higher than its batch model counterpart.

Green Information and Communications Technology (ICT) (MISHRA; YAZICI; MISHRA, 2012) is a hot trending topic in this millennia, along with the topic of the energy footprint (directly related to carbon footprint). More people are aware of resource consumption impacts on global warming (AKPAN; AKPAN, 2011) and the negative impacts of this phenomenon (NTANOS et al., 2015).

As pointed by Schwartz et al. (2019), most of the cutting edge development in machine learning leans more towards Red Artificial Intelligence (AI)[1], trying to find the best possible result for a given problem. Especially in a reasonably recent area such as data stream mining, many efforts go towards maximum accuracy in detriment to a healthy balance between accuracy and resource usage. Though with fewer studies, efficiency in data stream mining has important studies such as Martín, Lavesson and Grahn (2015)'s work.

An essential part of energy efficiency is measuring (Energy Magazine, 2018). Although vastly explored in ICT, measuring data stream mining is still a topic with much

[1]Red AI is the Artificial Intelligence that seeks to maximize the quality of the results. It does not try to be energy efficient. Green AI, on the other hand, tries to maximize the quality of the results, while having constraints on the efforts to obtain the result (see Section 3.1).

to explore. As seen in the work of García-Martín et al. (2019b), García-Martín et al. (2019a), various methods aim measuring data mining resource consumption and data stream-mining resource consumption. Green ICT applied to data stream mining is an area of knowledge with much to research and has been a trend to reduce energy consumption in data mining (MARTíN; BIFET; LAVESSON, 2020).

The main research question that is answered in this work is: "How to lessen the pain of measuring the energy consumption of data stream mining models during their development phase?" This question forks into other questions: "What is the impact of different data stream mining validation protocols in energy consumption?" and "What is the impact of having a concept drift in energy consumption?". Those questions are presented in Section 1.1. To answer these question, it is, therefore, necessary to bring forward a form of measuring energy consumption available for general use and that does not require custom build hardware. Therefore, the main goal of this work is to develop a plugin that measures energy consumption of data stream models. This plugin is an extension for the Massive Online Analysis (MOA) tool (BIFET et al., 2010) (see Section 2.6) that allows data stream researchers and practitioners to verify the power consumption over time.

There are many forms of measuring the consumption of a machine learning model. Most of the hardware solutions do measure the consumption of the entire machine, though García-Martín et al. (2019b) mentions ways to narrow down which processes consume energy in a server, defined by performance counters, Central Processing Unit (CPU) utilization, and other techniques. This approach may reflect the exact usage of an individual process in a multi-task environment. In a real-world scenario applied to data streams, one has to consider the idle period in which the model is not training or predicting. As shown by Barroso and Hölzle (2007), servers seldom use one hundred percent of their capacity, staying between ten and fifty percent of usage most of the time. For a data stream predictor used at a large scale application, one can assume the same, as it shares the same availability requirements that regular web servers have. It is not desirable to reach the server's full capacity or full inactive state, producing undesirable delays.

Due to the above reasons, this work's focus is the overall consumption with a dedicated server. Since most servers are not energy proportional (BARROSO; HÖLZLE, 2007), and the dynamic voltage range applies mainly to CPUs, this work takes static energy consumption (see Section 3.3) into consideration.

1.1 Research Questions

The main driving research question for this work is: "how to lessen the pain of measuring the energy consumption of data stream mining models during their development phase.?" To tackle the challenge of measuring energy consumption by a data stream model, we have the following questions that lie at the intersection of data stream mining and green computing.

- What is the impact of different data stream mining validation protocols in energy consumption?
- What is the impact of having a concept drift in energy consumption?

1.2 Objectives

Thus, with the research questions in mind, this work has the following objective:

- To provide a plugin for energy consumption measurement of data stream mining models that is available for the general audience of data stream mining.

The specific objectives include:

- Instantiate the proposed plugin for Massive Online Analysis (MOA) (see Section 2.6.1) tool, which controls and measures the model acquisition, training, and prediction phases.
- Compare the results given by the measurements, verifying if each change in the model will increase or decrease its energy consumption.
- Evaluate the impact on energy consumption when switching validation protocols and when concept drift occurs on a data stream model.

1.3 Financial support

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

1.4 Overview

This work has the following structure: Chapter 2 introduces the reader to concepts of Data Mining, Data Streams, Data Stream Mining, and Data Stream Mining tools. Chapter 3 introduces the concepts of Energy Consumption, Green ICT, and tools for Energy Consumption Measurement. Chapter 4 presents the current state of the art, and the gaps of knowledge that are tackled in this work. Chapter 5 presents the scientific method. Next, Chapter 6 brings forth the experimental setup and a description of the plugin developed to help the researcher in measuring the energy consumption. Chapter 7 presents the results of the experiments and Chapter 8 presents the conclusions based on the evidence, alongside limitations found and envisioned future works.

Chapter 2

Machine Learning in Data Streams

Arthur Lee Samuel coined the term machine learning in the 1950s. They appeared in the paper "Some Studies in Machine Learning Using the Game of Checkers." (SAMUEL, 1959) In this particular study, the author describes the method utilized to create a computer program capable of learning from experience. Back then, many wondered if those machines would acquire intelligence and think by themselves. And not too late, they started to be programmed to perform "intelligently."

> "Enough work has been done to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program." (SAMUEL, 1959)

In those early days, computers solved many problems that are considered difficult for humans, but relatively straightforward for computers, like problems that can be solved by a set of formal mathematical rules (GOODFELLOW; BENGIO; COURVILLE, 2016). However, it has proven that the real challenge for machine learning is to perform tasks that are easy for humans, such as intuitive tasks and pattern recognition, things a human performs without thinking much, and it almost feels automatic.

In today's understanding, machine learning is one category of artificial intelligence, a much broader area that includes most forms of computer knowledge. Machine learning is any form of artificial intelligence that can evolve with experience, e.g., data.

According to (BISHOP, 2006), machine learning is another facet of pattern recognition. An algorithm is said to be learning when it learns from experience. In other words, the task that produces a specific result with some accuracy can self adjust automatically utilizing sets of data. After each training session, the performance can vary, often producing more accurate results.

Data mining is the task focusing on discovering find different patterns, anomalies,

and correlations in sets of data, also referred to as knowledge discovery in databases. It accomplishes its results by making use of machine learning algorithms.

> "The field combines tools from statistics and artificial intelligence (such as neural networks and machine learning) with database management to analyze large digital collections, known as data sets." (CLIFTON, 2020)

Also, according to Moulet and Kodratoff (1995), Data Mining is a process in which one is concerned with "the construction or the completion of the correction of a knowledge base, by building rules, domain theories, discovering concepts, and so on, in order for the learner to be able to solve new problems better or to predict new situations."

One of the main goals of data mining is finding hidden patterns that cannot be easily seen or reasoned. The human mind excels at finding patterns, and trivial patterns are not necessarily the goal of data mining. There are many trivial examples one can think of, such as identifying that all women who gave birth were pregnant, or that all pregnant people were female.

The main goal of data mining is to find the real hidden gems inside all the raw data. One may or may not have heard of the story "beers and diapers." It is not for this study to investigate whether this story is real or not, but the story goes like this:

> Wal-Mart, the world's largest retailer, supposedly found out that there are certain times at which beer and diapers sell particularly well together – when on Friday evenings young men make a last dash to the supermarket to get beer and their wives call after them, "Pick up some diapers, too, honey!" (XIAO, 2014)

Although this story may not be real, it highlights one of the goals of data mining, which is finding meaningful hidden observations inside the data available.

One can think of traditional data mining as a batch process, in which all the available information is processed at once, outputting the model at the end of it.

In batch learning, the algorithm usually uses a static dataset, which is available to the algorithm in its entirety. At every learning cycle, all instances are processed (sometimes multiple times), producing an output model (GAMA, 2010).

Nonetheless, the data generation trend of growth is rising way faster than the speed in which new processors develop, or more memory becomes available. This trend has generated the need for new techniques to handle data, which also encompasses machine learning models.

Furthermore, traditional machine learning often does not take into consideration the time factor when dealing with the data available. It assumes all data comes from the same probability distribution and does not take into account how it changes over time. One possible solution is to utilize incremental and adaptive learning to build effective predictive models from data streams. Data streams are infinite sequences of data that may change over time. (BIFET; KIRKBY, 2009) When utilizing machine learning in data streams, new possibilities, and challenges appear. This chapter provides details on data streams and data stream mining, i.e., the scenario in which incremental and adaptive machine learning algorithms are applied to extract useful patterns from such type of data.

2.1 Data Streams

A stream is a continuous flow of liquid and gas. Moreover, the number of particles that go through that stream can be considered unlimited. A stream in computational terms is the continuous flow of computational elements. Per definition, a data stream is a sequence of encoded data used to represent information in transmission. Like streams, a data stream does not necessarily have an end, potentially having an infinite amount of samples. (MARGARA; RABL, 2019)

Intuitively, it is easy to imagine that the sheer number of samples will become insurmountable to fit in a single machine, and it is a challenge to process the data stream at once.

In the early days of computing, the amount of data generated was not much, as not many sensors were collecting this data, so all this data could fit in databases. Its growth was manageable by simply inputting in the existing database technologies and updating the data at regular intervals.

In recent decades, the growth of data gathering was astonishing, to a point where it became impossible to utilize the same database technique. The amount of data collected in a day surpasses, by far, the amount of data saved in a year thirty years ago, be it by more variables, more connected sensors, or other factors (REINSEL; GANTZ; RYDNING, 2018).

In comparison to databases, data streams do not provide the data at once, but they provide a variable sampling size.

Two data streams main characteristics are (GAMA, 2010): a potentially infinite source of data and evolution of the stream input over time.

Imagining a simple thought experiment: one has a computer that has ten units of memory and can process one unit per hour. If the data generation is one unit per year,

it will take ten years for the computer to reach its memory capacity. It will take ten hours to process this information, starting at one hour in the first year, and in the tenth year, it would take ten hours to process all the data units. In this scenario, it is possible to follow with a database strategy, as eventually, one would replace the hardware with a more powerful one, but time would be an ally. If the data generation is one unit per hour instead, in the first hour, the computer will utilize one hour to process it. When the second unit is available, the computer will take two hours to process. When the third unit becomes available, the computer is still working on the combination of the first and second samples.

The above example is how batch processing works. It processes all the samples, again and again, every time a new model is required. One argument is that one could space out the creation of the model, maybe once a day. Eventually, the processing time will be over that interval, reaching unimaginable levels. Moreover, at the mark of ten hours, the computer would run out of memory and to change hardware every ten hours or so seems impractical at best.

The fact that the impossibility to process the ever-increasing amount of data is upon us, and knowingly that the processing time is proportional to the amount of available data, there is a need for more efficient algorithms to process these streams of data.

The next chapter discusses some of the challenges of dealing with data streams when trying to extract knowledge from them, such as concept drift, memory consumption, and processing time.

2.2 Data Stream Mining

As of today, data generation is continuously increasing. Companies are collecting data about all aspects of the world, ranging from temperature measurements, to every click and move that a user performs when navigating in a web page.

A specific user's behavior while navigating a specific webpage throughout the entire webpage lifetime is a potentially infinite source of data. Imagining that one wants to analyze the user behavior to improve its user experience, it is seldom beneficial waiting for all the samples to start the analysis. For once, if the website lifetime is over, there will be no effect in improving that particular user experience, so it is imperative to analyze the data as they are generated for most of the cases. Because of the uncertainty about when the data will be complete, one has to treat it as an infinite stream of data. Another point to add is concerning the temporality of the data. The user behavior on the webpage may change over time, for many reasons, be it becoming more proficient at navigating

or changing its interests (WANG; WANG, 2012). The fact is that users do change their usual behavior, evolving as time passes.

Data stream mining is the application of data mining to data streams. When applying data mining concepts to data streams, it becomes clear that it is a challenge to cope with the sheer amount of data that an infinite source of data presents.

Along with the fact that traditional data mining processes data all at once, streamed data does change over time. Traditional data mining would treat any sample with the same approach. However, old behaviors may not be beneficial to predictive models' outcome, as only the most recent data may contain the essential data and patterns. Treating old and new data as differing only by its static probability would not only result in inaccurate models, as it would hinder the ability to extract meaningful information.

As outlined by Bifet and Kirkby (2009), there are certain requirements that must be met for data mining in streams:

- **Requirement #1**: Process an example at a time, and inspect it only once (at most);
- **Requirement #2**: Use a limited amount of memory;
- **Requirement #3**: Work in a limited amount of time;
- **Requirement #4**: Be ready to predict at any point; and
- **Requirement #5**: Be able to detect and adapt to concept drifts (see Section 2.3).

Figure 2.1 outlines one possible cycle in the algorithms that mine data streams. It begins by querying the stream, inspecting it exactly once. The received instance is then processed by the system, considering its limitations. Next, the predictive model is generated or updated. This model can be used to predict and evaluate how well the system performs.

2.3 Concept drift and Change detection

A peculiar facet of data streams is concept drift. When looking at batch machine learning, it is fair to assume that the model will fit the training set. If successful, it will have reasonably good accuracy when presented with new data that follows the same pattern as the training set. However, this assumes that the input data will not change because it is presented all at once. It also assumes there is no temporality in the data and that it does not evolve. When the result of the modeled phenomenon changes over time,

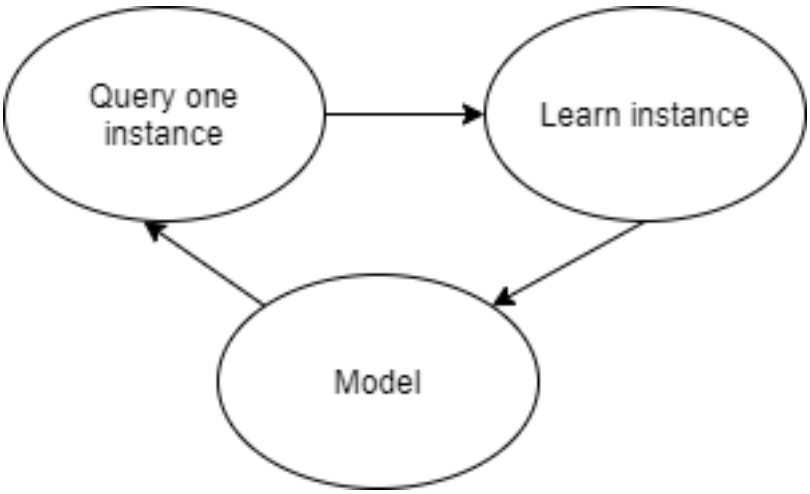

Figure 2.1: Data Stream Algorithm Cycle, as the model must always be ready to predict, queries one sample, processes it, and creates a model, iterating over it for the model's life.

we have a concept drift. In other words, its when the result of a specific input changes due to unknown or untracked factors (hidden context).

So, any "shift in the statistical properties of the data, more than what can be attributed to chance fluctuations" (BIFET et al., 2018) can be considered a concept drift. The concept drift changes can either be gradual or abrupt, with the main difference between them being the number of instances it takes for them to occur.

To exemplify a concept drift, one could take as an example depicted in Figure 2.2, where a beginner chess player that has learned to some extent one or two chess openings, when that particular player plays black and faces the King's Pawn Opening (1. e4), the usual response is (1 ... e5). With enough games, a model for that player will predict that

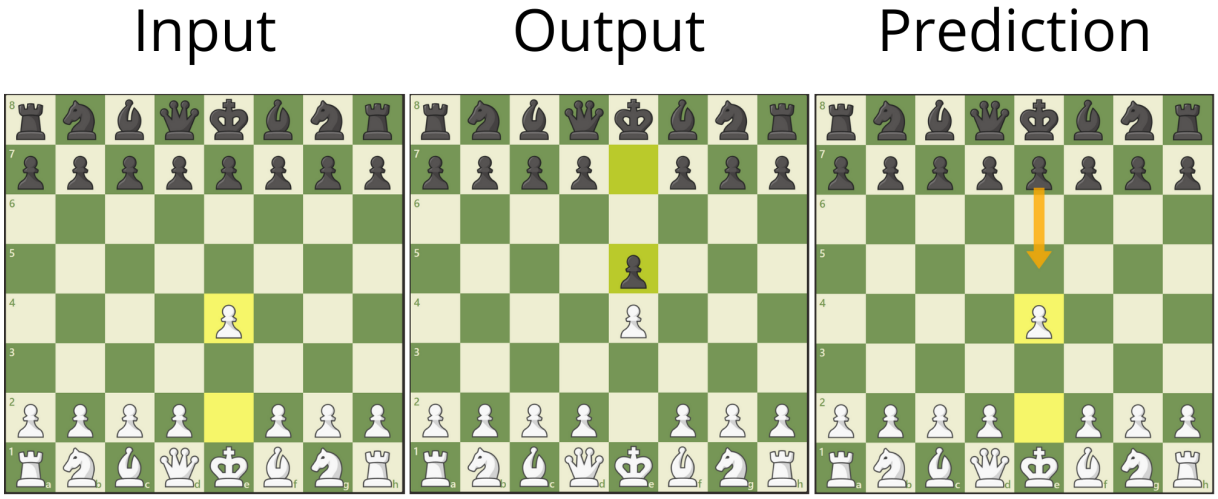

Figure 2.2: Beginner chess player's response to 1. e4

when faced with that arrange of pieces, the play will be (1 ... e5).

Later on, this player learns more openings, broadening his repertoire, and he learns how to play the Sicilian Opening (1. e4 c5). So the model predicts that the player will move (1 ... e5), and instead, the player will start responding with (1 ... c5), resulting in a concept drift (Figure 2.3).

There are different types of concept drift (LU et al., 2020) (see Figure 2.4):

- Sudden drift: It is when the change occurs abruptly. In this case, the transition period from the old output to the new output is minimal, i.e., it occurs between two subsequent instances.
- Gradual drift: the change between prior and subsequent posterior concepts changes within a time interval. During this interval, instances might be drawn from either prior and posterior concepts.
- Incremental drift: this type of drift is when the output changes to the new output with small increments over time.
- Reoccurring concepts: A concept might occur multiple times in a data stream. The change between concepts might be sudden or gradual, yet. This behavior is often observed in seasonal scenarios, where the relationship between predictive features and the output class is driven by timely and seasonal components.

Drift detection is one of the most tackled problems in data stream learning. The next section brings forward an outlook on drift detectors.

Figure 2.3: Player learns a new response to 1. e4

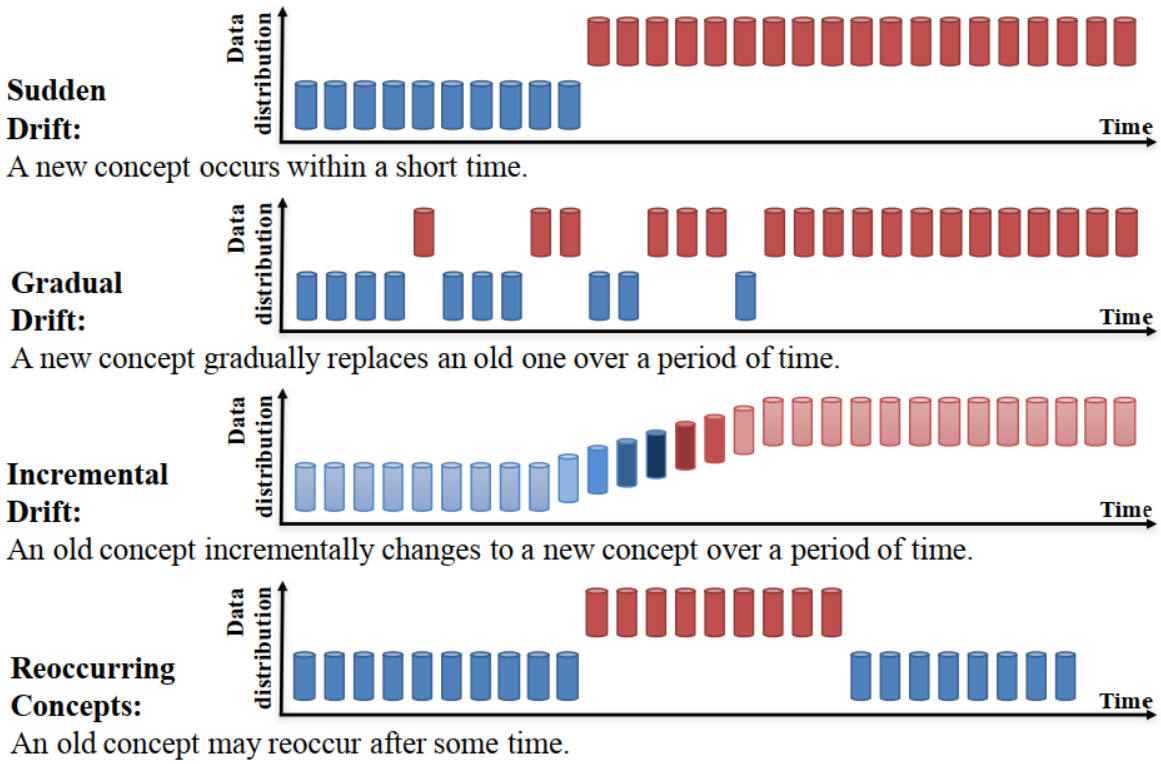

Figure 2.4: Examples of concept drift. Sudden, when a change occurs at once, gradual, in which there is a time with mixed results from the old and the new output, incremental when the change from the old output happens over small increments and reoccurring, when the output changes to an old concept from time to time, and goes back to the usual output. Image adapted from (LU et al., 2020).

2.3.1 Drift Detectors

Handling concept drift has become a popular research topic due to its multidisciplinary nature. In the past decade, a vast body of literature has been produced by the scientific community. The many surveys and overviews show the different methods developed to address drift, such as (GEMAQUE et al., 2020), (KHAMASSI et al., 2018), (GAMA et al., 2014), (BARROS; SANTOS, 2018), and (HU; KANTARDZIC; SETHI, 2020).

Drift detectors are classified into three categories, depending on the statistic tests they apply (LU et al., 2020):

- Error rate-based Drift Detection.
- Data Distribution-based Drift Detection.
- Multiple Hypothesis Test Drift Detection.

Each of the detector categories will be summarized next.

2.3.2 Error rate-based Drift Detection

As per Lu et al. (2020), this forms the largest category of algorithms. These track the changes in the online error rate of the base classifiers. When a change in the error rate is statistically significant, a drift alarm is triggered.

2.3.2.1 DDM - Drift Detection Method

One of the most referenced algorithm is the Drift Detection Method (DDM) (drift detection method) (GAMA et al., 2004). It pioneers defining warning level and drift level. DDM makes use of a landmark time window, that is, a window of time where the initial point is set and considers all samples acquired until the last received sample. When it evaluates new data, DDM detects if the overall online error has increased significantly in the landmark window. When the confidence level of the error reaches the warning level, DDM triggers the creation of a new learner to replace the old one. It still uses the old learner for predictions between the warning and the drift levels. It replaces the old learner when it reaches the drift level.

2.3.2.2 EDDM - Early Drift Detection Method

The Early Drift Detection Method (EDDM) detection method, proposed by (BAENA-GARCÍA et al., 2005) is an error-based concept drift detection method, much like the other methods listed in this Section 2.3. It possesses both warning and drifts trigger levels. Other approaches, such as LLDD, use the error to determine whether a drift has occurred or not based on the probability of the error occurring. The main characteristic of this detection method is its approach to the analysis of the errors. Instead of using the error rate, it uses the distance between two consecutive errors. The threshold for this method is thirty errors. It then gets the distance between each error and validates the distribution. When the distribution approaches the drift distribution, it signals that a drift occurred.

2.3.2.3 ADWIN - Learning from Time-Changing Data with Adaptive Windowing

Adaptive Windowing (ADWIN) (BIFET; GAVALDÁ, 2007) is another popular two-window error-based drift detector. Different from STEPD, ADWIN does not require the user to set the window size in advance.

ADWIN is a variable sliding window with variable length and change detector

(BIFET; GAVALDÁ, 2007). ADWIN keeps a history of recent examples, as long as they have the same probability. The length of the window grows as long as there are no changes in the input. ADWIN shrinks the window length whenever a change is detected, discarding old values that do not conform to the new probability. Essentially, it prunes the values that do not conform to the current probability of the input.

2.3.3 Data Distribution-based Drift Detection

Data distribution-based drift detection is considered the second largest category of drift detectors. These detectors use the distance between the distribution of historical data and new data to detect concept drifts. Since this category of algorithms detects the drift from the root cause (the underlying probability of the input), it is possible to determine the drift's time and location accurately. However, as indicated by Lu et al. (2020), these algorithms have reported a higher computational cost than the ones mentioned in Section 2.3.2. A general strategy adopted by this category is to utilize one fixed-size and one sliding window. If there is a significant difference between the distributions, it triggers a warning and starts to build a new model, and when it detects the drift, it replaces the model with the new one.

2.3.4 Multiple Hypothesis Test Drift Detection

The detectors that befall on this category use related techniques from Section 2.3.2 and Section 2.3.2. The originality, in this case, is the usage of multiple hypothesis tests to detect the drift. Those algorithms fall on two categories (LU et al., 2020):

- Parallel Multiple Hypothesis.
- Hierarchical Multiple Hypothesis.

In parallel multiple hypothesis, many hypotheses are working in parallel to decide if there is a drift or not. Depending on the structure of the algorithm, those hypotheses will address the detection differently. For instance, e-Detector (Ensemble of Detectors) (DU et al., 2015) proposes detecting an ensemble of different drift detectors. Then, following the ensemble logic, it decides whether it drifted or not.

Hierarchical Multiple Hypothesis, on the other hand, applies one test to detect drift, then validates in another test, essentially chaining the drift detection tests. Algorithms in this category include:

- e-Detector - Ensemble of Detectors (DU et al., 2015): The core concept behind an

ensemble of detectors is the same as an ensemble of classifiers: Some ensembles may be more robust at detecting an abrupt drift, while others may be stronger at detecting a slow drift. An ensemble of detectors will have the combined strength of them, thus stronger overall than each detector by itself.

- Three Layer Concept Drift (ZHANG et al., 2017): The three-layer concept drift is a detector applied to text data streams. It works by detecting three types of drift: a change in the label space, a change in the feature space, and a change in the mapping relationship between features and labels.

2.3.5 Energy Efficiency of Drift Detection

At its base, most drift detectors work in a two-step manner. The first step happens when it detects a deviation in the pattern up to a certain threshold (it signals a warning), and the next step is when it determines the drift is above the drift threshold when it signals a drift. Despite differences on how detection takes place, the goal is to reset or change the predictive model accordingly so that it reflects the reality depicted in the data distribution more accurately. Each learner copes with changes differently, but most models require either pruning of parts that do not conform to the new paradigm, e.g., decision trees, while the remainder discards the model entirely.

A recent approach used to combine drift detectors and learning algorithms, is to create a new model whenever a warning is issued. This mechanism is called "background learning" (GOMES et al., 2017). This new model will learn in parallel to the old one (still used for predictions) while the drift detector does not meet the drift threshold. After the drift threshold is reached, the old model is discarded and it is replaced with the new one.

When a new model starts training in the warning period, it will utilize resources from the machine it is running on, i.e., CPU time, memory, and storage space. This usage will increase as time goes on, increasing the model's overall consumption, accounting for all of the present models, the detection algorithms, and the new models in training.

When it finally reaches the drift threshold, the discarded model may have a different resource footprint as the old model, smaller or larger. However, it is fair to assume it will utilize a similar amount of resources, mainly because it uses the same technique and technology.

In essence, any drift detector will utilize some resources to detect the drift atop the model's resources. It also requires some algorithm that deals with the drift. This algorithm also allocates some resources for itself, and finally, the execution when it finds a drift. The transition between models also requires resources, so, when there are frequent

drifts in the model, it is fair to assume it will utilize more resources than a stationary model.

2.4 Data Stream Classifiers

A data stream classifier is a classifier adapted to work with data streams (GAMA, 2010). Many known data mining strategies need to adapt when dealing with data streams because of their innate characteristics (see Section 2.1).

Moreover, one significant difference from the traditional classification is how training takes place in data stream learning. Since data streams are potentially infinite, it is impossible only to train if all the data is available. Also, due to its temporality characteristic, a data stream classifier has to train as samples arrive.

Some of the data stream classifier's characteristics that it needs to follow include:

- The data stream classifier must carry just enough information to produce a model. In other words, data stream classifiers should not save all the received instances and based all the data received, but instead, should have some logic that either keeps a recent history of data, a summary of the data or both (BIFET; KIRKBY, 2009).
- At any given time, the model must reflect the classifier's current state and be ready to predict on demand (BIFET; KIRKBY, 2009).

The following sections describe data stream classifiers used in the remainder of this work. For an exhaustive list on existing works on data stream classification, the interested reader is referred to the works of (ZHENG et al., 2020; MEHMOOD; ANEES, 2020).

2.4.1 Naive Bayes

A Naive Bayes classifier (WEBB, 2010) has its roots in Bayes theorem (1763). Bayes and Price had a remarkable conversation that led to the theorem of Bayes (equation 2.1):

$$P(y|X) = P(y) \times \frac{P(X|y)}{P(X)} \tag{2.1}$$

In short, this theorem denotes that it is possible to find the probability of y given X, knowing the probability of X given y and the probability of X.

In the Naive Bayes classifier, the learner naively assumes that the input features are independent, such as depicted in Equations 2.2 and 2.3.

$$P(x_i \cap x_j) = P(x_i) \times P(x_j) \quad | \; i \neq j \tag{2.2}$$

$$P(x_i) = P(x_i|x_j) \quad | \; i \neq j \tag{2.3}$$

More specifically, Equation 2.3 can be used due to the independence premise between the inputs, and it calculates the output probability of a given class as the multiplication of the probabilities of all outputs, as depicted in Equation 2.4.

$$P(y|X) = \frac{1}{P(X)} \times P(y) \times \prod_{i=1}^{n} P(x_i|y) \tag{2.4}$$

Because Naive Bayes learners do not require the input of complicated hyper-parameters, it is one of the easiest and widely used classifiers available. (MUKHERJEE et al., 2019).

Naive Bayes learners are easy to implement in a data stream scenario. All one needs to calculate the new probability is the number of instances of a class that appeared for a particular input variable, since it only calculates $P(X)$ and $P(x_i|y)$ probabilities. Since it assumes the input probabilities are independent (Equation 2.3), it is easy to calculate $P(x_i|y)$.

For each new input, the learner updates the probabilities solely based on the count and does not store the actual input. This characteristic gives this type of learner an excellent edge in computational time, as it is constant in a data stream scenario, in both memory and processing times. It also means that the model will not grow as the number of instances increases, as the only change is the fluctuation of probabilities.

2.4.2 k-Nearest Neighbors

k-Nearest Neighbors (kNN) (KEOGH, 2010), is another supervised learning technique. Its name comes from Altman (1992). It consists of classifying or regressing a sample based on its nearest similars. kNN works by asking the neighbors that mostly resemble the sample, e.g., but measuring the distance between the sample and a historical one; what class they are, and utilizing a voting system (weighted or not), it decides what class the query sample belongs to. One such way to compute the distance is to imagine the attributes of the instance are the coordinates in a multidimensional space. There are many metrics utilized to verify the neighbors, though two are very straightforward and easy to reason: the Manhattan and Euclidean distances.

The Manhattan distance is the simple sum of the difference in each axis, given by equation 2.5. In the equation, a and b are the two points to measure the distance, i

represents the axis, and n_d is the number of dimensions of the measured space.

$$d_{\text{Manhattan}} = \sum_{i=1}^{n_d} |a_i - b_i| \tag{2.5}$$

The Euclidean distance is the distance in a straight line from A to B, and it is given by Equation 2.6. It uses the same notation as Equation 2.5.

$$d_{\text{Euclidian}} = \sqrt{\sum_{i=1}^{n_d} (a_i - b_i)^2} \tag{2.6}$$

kNN has a great property to be used in data streams. It does not require any complex learning process, so updating it with a new sample requires constant time. Classical kNN increases the memory imprint linearly as it adds more samples to the model (it just remembers the sample, and no processing is required). However, there are many drawbacks to using kNN in a data stream scenario. For once, it cannot handle massive datasets, as it runs out of memory. Imagine a scenario with a vast amount of samples, namely n_s. Given each sample occupies the same amount of memory m_s, the memory consumption is $n_s \times m_s$. If n_s is massive, then the result is even more massive. It also requires more computational time as it grows, as for a classifier with n_s stored samples, it requires the computation of n_s distances. Furthermore, each distance computational cost increases linearly with the increase in dimensionality n_d, so the minimum amount of calculations required every time a new sample arrives is $n_s \times m_s \times n_d$.

To cope with an infinitely growing dataset, kNN must prevent the buffered dataset from exhausting all the available resources. One possible solution to limit the buffer size is the utilization of a sliding window. The sliding window can limit the number of instances by the maximum number of instances or by a time frame. If the window has not reached its space or time limit, it adds the new instance to the buffer. If this limit is reached, it buffers the new instance and discards the oldest data.

Due to a possibly massive number of instances for classification, kNN must come with a classification solution to compare values. Since calculating the distance between all instances in a multi-dimensional space is very time consuming, one possible solution is the utilization of K-Dimensional (K-D) trees (BENTLEY, 1975). K-D Tree is a binary tree, whose node values are points in a multi-dimensional space. The basic idea of a K-D Tree is to create sub-planes aligned with the axis (so precisely one type of alignment for each dimension) and successively split the points based on one of the alignments. For example, a three-dimensional space would first have a plane aligned with the x-axis, then a plane aligned with the y-axis, and finally, a plane aligned with the z-axis. It would split

instances in two for each level of the tree based on whether they are to the left or to the right of the plane that intersects the node value. Then, it would take a plane aligned to the next axis for the next level, split the instances successively to the left and right.

2.4.3 Decision Trees, Hoeffding Trees and VFDT

One of the main families of classifiers utilized in machine learning is decision trees. A decision tree is a structure that the values must travel in a path, performing many tests. Each decision tree has a root that analyzes one of the attributes of the input. Depending on the result of the analysis, it takes a specific path. This path may lead to another split (decision) node or a leaf. If reaching a split node, it conducts another analysis, which leads towards another subpath in the tree. If it reaches a leaf, it contains the class of the input.

When training a decision tree, there are algorithms to decide what attribute each node will test. There are many approaches to this, but not limited to: Iterative Dichotomiser 3 (ID3) (QUINLAN, 1986), C4.5 (SALZBERG, 1994), CART (BREIMAN, 1993), CHAID (KASS, 1980), amongst others. For each of the aforementioned algorithms, one of the main concerns is the selection of the attribute for each node. All of the traditional algorithms make use of some statistic that defines each attribute, how it impacts the result, and how present it is in the population. Having the entire population available to the training algorithm from the start makes this question irrelevant since one can perform the statistical analysis for each of the attributes. Though decision trees are easy to create and maintain in a batch scenario, it may not be straightforward when dealing with a data stream scenario. A classical approach would recreate the tree every new sample, and imagining that a data stream is continuously receiving new training data, it is possible to extrapolate many of the pitfalls, such as increased processing time in the order of $O(n_{samples} \times n_{features} \times log(n))$ every time a sample is processed, and $O(n_{samples}!)$ overall. Other problems mentioned by Domingos and Hulten (2000) include:

- Not all examples will fit into memory;
- No guarantee that the model learned sequentially will be similar to the one generated by a batch learner;
- High sensitivity to example order; and
- High cost and low efficiency.

To overcome these trade-offs, the same author proposed Hoeffding trees. Hoeffding

trees have a particular characteristic: the time to learn per example is, in the worst-case, proportional to the number of attributes (DOMINGOS; HULTEN, 2000). That is, as the number of inputs increases, the time to evolve the tree from more samples remains uniform.

Very fast decision trees work by constructing decision trees using constant time per sample. The decision on whether a split is performed or not is assessed as soon as statistics about n samples are stored in a leaf node, a parameter referred to as 'grace period' (HAN; KAMBER; PEI, 2011; BARDDAL, 2019).

The Hoeffding bound (the upper bound) is the probability that a specific attribute will deviate from its real mean by an absolute value ϵ (HOEFFDING, 1963)[1]. The Hoeffding bound states that the when observing n samples, their true true mean $\overline{r}$ is at least $(\overline{r} - \epsilon)$ with probability $(1 - \delta)$. Hoeffding trees calculate the Hoeffding bound of each attribute. It uses to compare the two best-ranked features according to the chosen heuristic. If the gain is higher than the Hoeffding bound, the node splits.

$$\epsilon = \sqrt{\frac{R^2 \ln \frac{1}{\delta}}{2n}} \tag{2.7}$$

Domingos and Hulten (2000) also proposes Very Fast Decision Tree (VFDT) (Very Fast Decision Trees learner), a type of decision tree based on Hoeffding trees. So, the final proposition for VFDT has many refinements over Hoeffding Trees, namely ties, G computation, memory, handling of poor attributes, better initialization and rescans[2].

It is also important to note that VFDT can hold processing several samples until it meets a certain threshold. It also calculates the number of samples required to get within the desired probability, given the acceptable ϵ.

Though the authors refer to Hoeffding Trees and VFDT as different algorithms, the community uses their names interchangeably, referring to Hoeffding Trees when they meant VFDT. Though Hoeffding Trees are significant improvements over decision trees when it comes to data streams, it is not without drawbacks. One major drawback when utilizing Hoeffding Trees is that the tree is expected to grow indefinitely as new data arrives, as noted by experiments conducted in (KARAX; MALUCELLI; BARDDAL, 2019). Due to the ever-growing size, any experiment projected to run forever poses a considerable problem. If working with limited resources, there will be a point where the experiment will come to a halt. If the experiment automatically allocates more resources as needed, it will eventually consume many resources, probably more than anticipated. In terms of

[1]Note: ϵ is used by Domingos and Hulten (2000), thus will be referred as ϵ in this essay.
[2]Note: The improvements for VFDT can be further read in (DOMINGOS; HULTEN, 2000).

efficiency, this is far from ideal, as the main interest here is to reduce energy usage, not increase it.

2.4.4 Bagging and its variants

Bagging (BREIMAN, 1996) is a supervised learning algorithm that generates multiple models and combines them to create a stronger one. This aggregation of models is an ensemble. Bagging is used because of its theoretical accuracy and proved to increase the performance of the individual base models (OZA, 2005). The term bagging is an acronym for Bootstrap Aggregating. The main idea behind bagging is to create multiple learners with the same learning algorithms but fed with different data (instances) from the same underlying distribution (BREIMAN, 1996).

The bootstrap aggregating method's main characteristic is creating multiple subsets of instances, one for each learner. Each subset is created with a "bootstrap" sample with replacement from the inputs (BREIMAN, 1996). Each subset is then given to a learner for training. That will form an ensemble of learners. When a new sample is made available for testing, it is given to all models, that will produce a result. The total result is the average or majority vote obtained from all learner's results.

Online bagging, proposed by Oza (2005), uses bagging to generate an ensemble from a data stream. It behaves similarly as the batch learner version, in the sense that it creates multiple learners and present random samples for each in for training. The main difference lies in many facts: because the samples come one at a time, they must be presented to the learners differently. As each training example $instance = (x, y)$ is presented to the ensemble algorithm, for each base model, choose the example $K \approx Poisson(\lambda = 1)$ times and update the base model accordingly using the online base model learning algorithm L_o. Consequently, each example will be chosen roughly $K \approx Poisson(\lambda = 1)$ times the number of learners available and distributed between the learners. Each learner that received the sample then updates itself.

When classifying a sample, online bagging does the same as batch bagging: it presents the instance to all models. Each model then presents a result that aggregates into a vote. The final result is the class that got more votes in case of a classification, or the mean of the results in case of regression.

To tackle concept drifting data streams, authors in (BIFET et al., 2009) proposed a combination of Online Bagging with drift detectors called ADWIN Bagging. The main idea behind ADWIN Bagging is to trigger the reset of classifiers when drifts are detected. Each learner in the ensemble is associated with an ADWIN drift detector (BIFET;

GAVALDÁ, 2007), and when a drift is signaled by a detector, its corresponding learner is reset (BIFET et al., 2009).

Leveraging bagging is another improvement to Online bagging, proposed by Bifet, Holmes and Pfahringer (2010). The paper proposes two randomization improvements: increasing resampling and the usage of output detection codes. The increase in resampling is a modification to the $Poisson(\lambda = 1)$ distribution in the original algorithm. They changed to $Poisson(\lambda = 6)$, and according to the author, it is "increasing the diversity of the weights and modifying the input space of the classifiers inside the ensemble. However, the optimal value of λ may be different for each dataset." The output detection improvement in leveraging bagging adds randomization to the output of the ensemble and is relevant for non-binary classification problems. It builds a binary string of length n, and n classifiers, and each class is assigned a random binary code of length n. Each classifier is assigned to learn one bit. The full output is a binary code, and the resulting class is the one that is more similar to the output. According to the author, this approach improves diversity because each classifier tries to learn a different function, whereas, in standard ensemble methods, all classifiers try to learn the same function.

2.4.5 Adaptive Random Forests

A random forest (BREIMAN, 2001) is an ensemble-based classifier, whose popularity has grown over the past years due to many advantages it has over other methods. Random forests have higher theoretical optimal accuracy when compared to individual methods. Random forests have a high learning performance and can cope with a wide variety of input shapes and forms (thus requiring very little input preparation and little hyper-parameter tuning). Because of that, Random forests are among the most widely used algorithms for batch learning. This phenomenon occurs because a random forest possesses a high learning performance, demands little input preparation and low hyper-parameter tuning (GOMES et al., 2017).

Like bagging and boosting, random forests are an aggregation of multiple classifiers. Focusing on data stream scenarios, the traditional algorithm for random forests is not applicable, so authors in Gomes et al. (2017) proposed Adaptive random forests to address this gap. The main idea of the adaptive random forest comes from Online Bagging (OZA, 2005). Thus, it uses $Poisson(\lambda = 6)$ whenever a new sample comes to the learner. It also utilizes a drift detection algorithm, not specified, though the authors tested with ADWIN (BIFET; GAVALDÁ, 2007) and Page-Hinkley Test (PHT) (MOUSS et al., 2004).

As the main characteristics of Adaptive Random Forests, one can name the eval-

uation of features based on a random selection and background learning.

It is interesting to note that background learning is a technique that generates another tree when it detects the current model that differs from what is being presented. Therefore, it is very relevant in the energy consumption context.

2.5 Validation

There are various validation methods for batch learning, such as holdout, bootstrapping, and cross-validation, to name a few (TANTITHAMTHAVORN et al., 2017). Most of the methods used for batch learning rely on the fact that the data is available all at once. However, data streams availability is different, and instances are not available at the start, nor are they sure to come regularly.

Thus, a data stream scenario may require different methods to validate the models. Two of the many available methods for validating data stream models are prequential and interleaved chunks test-then-train (mini-batch learning).

2.5.1 Prequential

The Prequential validation method, proposed by Gama, Sebastião and Rodrigues (2013), utilizes the idea that it first tests the model for each sample to check its accuracy and then train with the instance, improving its accuracy. As mentioned by the author, the advantage of prequential is that the model tests then trains with samples it had seen before (see Figure 2.5), requiring no holdout set, fully utilizing the available data. It is essential to mention that prequential does assume the availability of the labels before the next sample arrival. This assumption might not be possible in a real-world scenario. The prequential method possess both a sliding window and fading factor to account for old data with smaller weights during assessment.

2.5.2 Interleaved Chunks

The interleaved chunks validation method (BIFET et al., 2018) uses the same rationale as prequential in the sense that instances are first used for testing and later for training. The difference between prequential and interleaved chunks is that the data stream mining process takes place using data chunks instead of on an instance basis (see Figure 2.6). Interleaved chunks is more realistic than prequential as it does not depend on the prompt availability of instances' labels right after features are presented to the

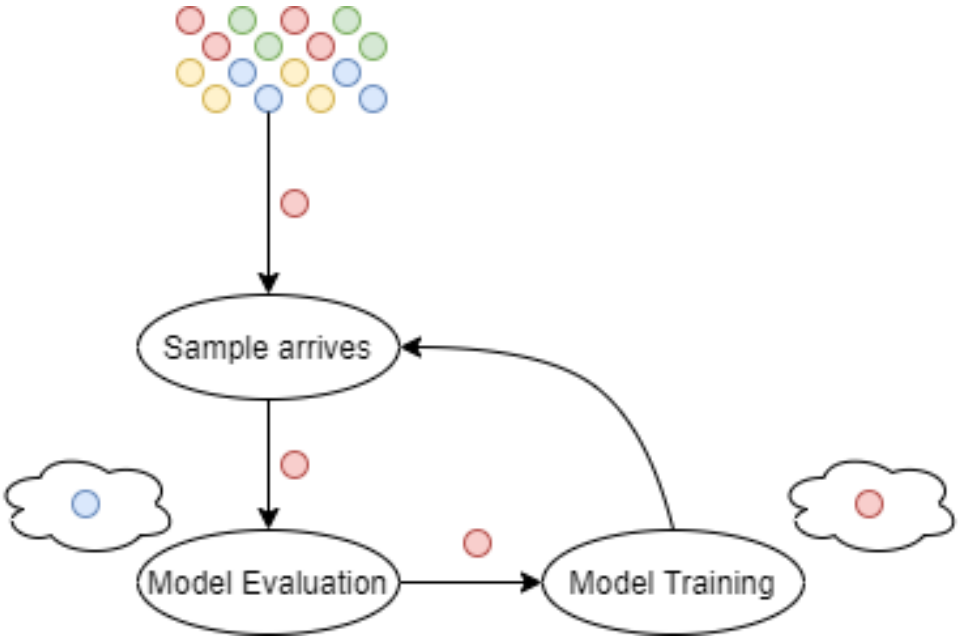

Figure 2.5: The prequential validation method loops indefinitely in a cycle, receiving one instance, testing, and training. In this figure, each circle corresponds to a sample, and the color represents a class.

learner.

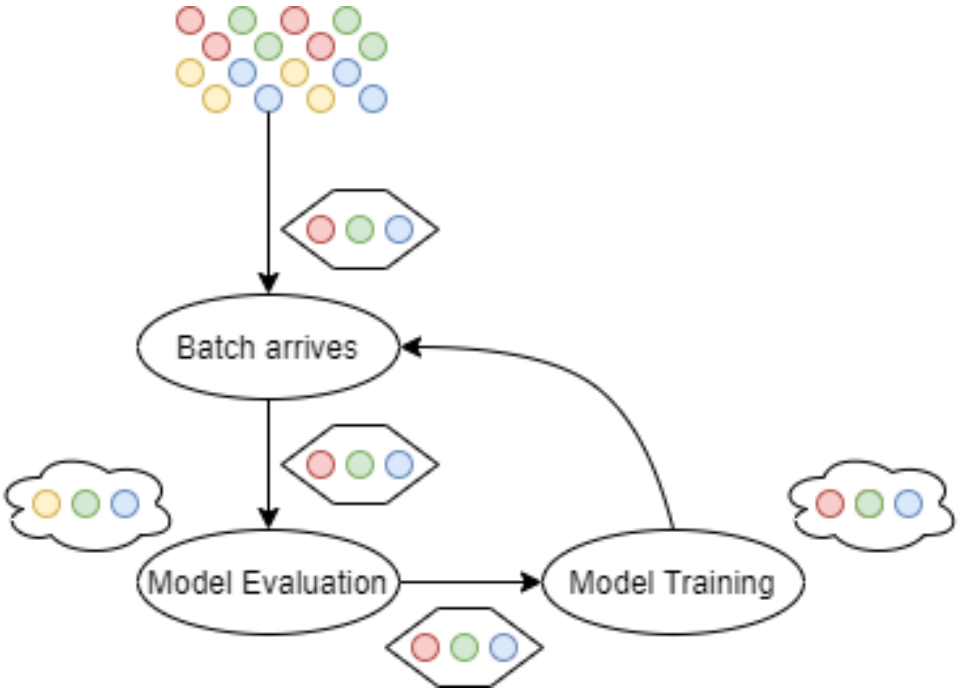

Figure 2.6: This figure demonstrates how interleaved chunks work, as it is almost like prequential, except it works in batches, depicted by the hexagons.

2.6 Tools for Data Stream Mining

Even though most of the data stream mining strategies and algorithms are either the same or derived from the ones utilized by batch learning (regular data mining), there are many nuances and differences that one addresses differently in a data stream scenario. Two of those differences are incremental learning and concept drift. The tools used in batch learning cannot handle those traits. While it is possible to code each algorithm individually for each problem, it is not practical during exploratory analysis.

There are a handful of tools for data stream mining, having at the time of the writing of this work, MOA (BIFET et al., 2010) and scikit-multiflow (MONTIEL et al., 2018) being the ones that are the most commonly known.

2.6.1 MOA

MOA (BIFET et al., 2010) is a software environment prepared to run experiments and implement algorithms in a data stream scenario. It already possesses several online and offline methods and evaluation tools. MOA is inspired by another software Waikato Environment for Knowledge Analysis (WEKA) (FRANK; HALL; WITTEN, 2016), and it is compatible with it. MOA is written in Java, and most of its implementations and extensions are also in Java or one of its compatible programming languages.

The methods available when the original paper was published were boosting, bagging, and Hoeffding Trees, with and without Naive Bayes classifiers at the leaves. Yet, MOA is constantly updated and at the time of the writing of this essay, it encompasses multiple learning and evaluation algorithms.

2.6.2 Scikit-multiflow

According to Montiel et al. (2018), scikit-multiflow is a Python framework prepared to run experiments and implement algorithms in a data stream scenario, similar to MOA (Section 2.6.1).

At the time of writing the essay (MONTIEL et al., 2018), scikit-multiflow encompasses many classification, regression, and clustering algorithms, including support vector machines, random forest, gradient boosting, k-means, DBSCAN.

Scikit-multiflow takes its inspiration from scikit-learn (FABIAN et al., 2011) and MOA. Much like its batch counterpart, scikit-multiflow is written in Python and can be extended using Python.

2.6.3 Remarks

In this work, it was decided to use MOA because it is being used by the community the longest. Another factor that impacted the decision is that Scikit-Multiflow contains experimental code that may not be ready for production environments.

2.7 Concluding remarks

Data stream mining is an exciting branch of data mining that has many pitfalls and difficulties. The state-of-the-art learners can overcome those challenges, but not without cost. Some learners may utilize many resources to produce a high-fidelity model, while others produce satisfactory results, utilizing a fraction of the processing.

Also, not only the best result may not be significantly better than other results, but it may have a much higher energy cost to produce and utilize.

The next chapter introduces Green ICT and its importance to data stream mining. These concepts will draw a clear picture of the importance of energy efficiency and how it affects the utilization of data stream mining models throughout its entire lifecycle.

Chapter 3

Energy Efficiency

Energy efficiency is a buzzword and a trend. In all areas that are targeted, it can mean different themes and applications. In this work, energy efficiency takes the definition of how much energy a model utilizes to achieve a goal. It is both a key concept and a final goal in this work.

As outlined by Mishra, Yazici and Mishra (2012), Green ICT refers to ICT (information technology) concerned with environmental sustainability. It is a collection of best practices and techniques to promote and develop energy efficiency and reduce waste. Therefore, Green ICT is the discipline that studies the full life cycle of computing, from the initial manufacture, though the entire usage and its final disposal, aiming to improve the overall resource footprint. Concerning usage, Green ICT concerns with the computers' energy footprint, its peripherals, and the required infrastructure for the proper functioning.

The world population is growing increasingly more aware of the available natural resources depletion. The need for efficiency and reduction of consumption is a trend, and it is both an economic and sociological necessity. Many studies point out that the increasing energy consumption growth is a trend, and the principles of green ICT should be followed (SCHWARTZ et al., 2019), (SCARFO, 2013), (PETROV; PETROVA, 2019), (ENOKIDO; AIKEBAIER; TAKIZAWA, 2010), amongst many others.

Energy awareness is another buzzword that has grown in popularity in recent studies. In this work, when mentioning energy awareness, we refer to the knowledge of energy usage that pertains to machine learning models. We also refer to energy awareness as the researcher's knowledge of the energy usage of a model, both during its training and prediction phases.

This chapter introduces Green AI (Artificial Intelligence) (SCHWARTZ et al., 2019), concepts of CPU time, static power, adds discussions on data stream mining, and

introduces the concept of horizontal and vertical scaling.

3.1 Green AI x Red AI

It is crucial to start this section with a disclaimer: by no means, this work criticizes the called Red AI (Red Artificial Intelligence) and its search for maximum accuracy, but instead, it brings awareness to the fact that energy efficiency is necessary.

Green AI is a term coined by Schwartz et al. (2019), and it describes the application of Green IT and its concepts to artificial intelligence. The aforementioned paper proposes utilizing efficiency as a more common criteria for evaluating AI models, alongside accuracy and related measures. Schwarts compared Red AI against Green AI (Green Artificial Intelligence). Red AI is the state of the art accuracy driven AI that consumes ever-increasing training hours in search of smaller gains (MAHAJAN et al., 2018).

In the paper by Mahajan et al. (2018), which refers to a linear classifier's accuracy when trained with ImageNet 1k, 5k, 9k (DENG et al., 2009), and CUB2011 (WAH et al., 2011), the authors outline the diminishing returns in accuracy when increasing the number of training instances.

Mahajan et al. (2018) utilizes various methods to train neural networks and verify its accuracy on hashtag prediction. On each graph (IN-1k, IN-5k, and IN-9k - The ImageNet datasets), the number of instances increases exponentially[1]. Consequently, the accuracy gains are smaller for each new instance, as it requires ten times more instances for a significant accuracy increase[2]. As mentioned by Schwartz et al. (2019), the amount of computing power used to train deep learning models has increased 300,000 times in 6 years, the difference between the number of instances used for AlexNet and AlphaGo Zero).

Another astounding project is Leela Chess Zero[3]. Leela Chess Zero has managed to win two computer chess championships utilizing a self-trained neural network. Leela Chess Zero is a community-driven open-source chess engine that utilized the same approach used by AlphaZero (SILVER et al., 2017). The team behind Leela started the same training the AlphaZero team has done, however, it is backed by a global community that works as its training machine, while AlphaZero relies on Google's supercomputer as its backbone. It took several months of worldwide training for Leela to compete at the

[1]The number of instances axis is logarithmic.

[2]The increase of accuracy for ImageNet-1k between 10^7 instances and 10^8 is less than 10% in all scenarios. This trend also follows from 10^8 instances to 10^9 instances.

[3]https://lczero.org/

premium league of both TCEC (Top Chess Engine Championship)[4] and CCC (Computer Chess Championship)[5]. It is unknown the amount of resources spent on Leela Chess Zero to achieve these results. However, a simple analysis considers that it spent several times more resources than AlphaZero did (AlphaZero trained for eight hours, while LeelaChess Zero trained for months distributed in many computers and is still training at the time of writing this work).

3.2 CPU time versus idle process in Data Stream Mining

When measuring the efficiency of a process in the context of energy efficiency, most efforts fall in the optimization category. Imagining, as an example, a model in the MOA framework[6] that takes 10ms to update a predictive model using a single labeled example, will utilize a total of 10,000s to train one million instances. If this model is optimized to update its model in 5ms, it will cut down the total training time by half. The unit used to verify how much time has a process used from the CPU is called CPU Time (GNU, 2020), and the CPU time is directly proportional to the energy consumed by the system.

Optimizing a data stream mining process making it faster will most likely reduce the overall consumption of the process, as it will utilize less CPU time during training or prediction. However, a data stream mining model running in production (versus a controlled environment like a lab or testbed) will not always be in use. There may be times when there is no input data to process or no predictions to be made. There may be two opposite assumptions made in this case. The first one is the machine in which the model is running will cease to spend resources with it and will be available to process any other process that requires attention. In this case, the idle energy signature of the model would theoretically be zero. The second assumption is that the model stays in memory, even not when in use. It occupies an ever-increasing memory amount from the machine, with an increased energy footprint (see Section 3.4). Another process may utilize the machine's idle computational power to perform some work. However, because some of its available memory is with the data stream model, the machine's efficiency will not be the best. There will be cases that dedicated hardware is assigned to the model to increase its efficiency, so all the static power footprint (see Section 3.3) is indeed generated by the model. Most of the time, server machines are not in a completely inactive state. Due to

[4]https://tcec-chess.com/
[5]https://www.chess.com/computer-chess-championship
[6]https://github.com/Waikato/moa

underlying processes or watches, a minimal amount of activity may generate an expressive consumption, even doing minimal work. In these many scenarios, the total static power is due to the model running on the machine.

One of the primary energy consumers inside data centers and computers, in general, is what is called static power.

3.3 Static Power, Energy Leakage and Idle Energy Consumption

Static power is, in essence, energy leakage at the transistor level. The electronics (transistors, capacitors, and others) inside computer components do flow current when in an idle state due to electrical characteristics. This phenomenon is called static power (JACOB; NG; WANG, 2008).

When components are active, meaning the electronics inside them are transitioning from state one and zero, based on logic, those transitions incur dynamic power, which is a current flowing, and this current flow is what we call power consumption.

Though, when components are in an idle state, and there is no activity occurring, electrical leakage occurs due to the current components' technology. That ultimately means that each part of the computer will have some energy footprint when idle. Because each component construction is different, having different brands and models with different purposes, there is a wide range of static power footprints. For example, while some notebooks CPUs focused on energy saving possess a dynamic voltage range, allowing it to diminish its consumption when almost idle, other components, such as high-end performance-focused CPUs, are always consuming one hundred percent of its peak consumption to provide peak performance when required (BARROSO; HÖLZLE, 2007).

While most components possess some shape or form of energy-saving, some strategies are not feasible in the most common scenarios. For example, spinning hard drives can go into a complete idle state and consume absolutely no power. However, when required to start its activity, there is a hefty power-on energy footprint, alongside a long wait period before it is functional.

Other components, like memories, require energy to maintain what is in the memory. Most notably, DRAM (LAPLANTE et al., 2018) require a periodic refresh to prevent losing its contents (memory refresh, (LAPLANTE et al., 2018)).

3.4 CPU Energy Footprint versus Peripherals Energy Footprint

The energy consumption by the CPU of a server, contrary to popular belief, does not have the highest energy footprint in a server environment, especially when the server is not running at full capacity (BARROSO; HÖLZLE, 2007; OMONDI; ATEYA, 2019). Other components are far less energy efficient than the CPUs. For example, memory needs electricity to maintain random data, and hard drives require electricity due to their rotations per minute. The option for low consumption at lower utilization levels is when the component is inactive. The inactive state for these components incurs a hefty penalty when transitioning from the inactive state to the active state, and memory cannot store any information at all when in an inactive state (BARROSO; HÖLZLE, 2007).

The energy consumption must be measured not only in the computer hardware but the entire infrastructure required for the server to run, such as cooling, networking and power stations, amongst other pieces of equipment (MISHRA; YAZICI; MISHRA, 2012).

Barroso and Hölzle (2007) points out that most servers have a load between ten and fifty percent of their maximum utilization. This fact leads to two remarks that are most relevant for this work: when the server is under minimal load, but not inactive, most of its components will have a different static power footprint (Section 3.3), and the server is most efficient when it reaches one hundred percent utilization. Also noted by the authors, an energy efficient server would consume fifty percent of its nominal power at zero utilization, but if the server is not built for efficiency, its energy footprint at zero utilization may be much higher.

This CPU versus peripherals' energy footprint is a significant factor when doing computation at scale[7], and it is necessary to increase the number of resources available to the model. (see the next section, i.e., Section 3.5). When scaling the model, there are many possibilities, such as horizontal and vertical scaling, and they behave quite differently in terms of energy consumption.

3.5 Vertical and Horizontal Scaling

Vertical and horizontal scaling is also known as scale-up and scale-out, respectively (MICHAEL et al., 2007). In short, vertical scaling is when an application is deployed on

[7] At scale refers to adjusting the number of resources to the amount necessary to solve the problem.

a single large server, and its growth happens increasing memory, swapping the processors, and adding more disks to the same server. On the other hand, horizontal scaling is when the application lives in small interconnected servers, and its growth happens by adding more servers. There are many different aspects with both approaches, as they require specific architectures, technologies, and topologies. In terms of raw cost, vertical scaling tends to be more expensive than horizontal scaling because of hardware issues. The motherboards can only handle a certain amount of memory, and processors get exponentially more expensive according to their speed. Whereas for horizontal scaling, each server added to the cluster has a fixed cost, having a linear increase in cost as it scales-out.

One can see an example of vertical scaling in Figure 3.1. The increase in the amount of installed DRAM from 32 gigabytes to 128 gigabytes increases the computational stack's processing potential, without increasing the number of machines to work. Other vertical scaling strategies exist, such as changing any part of the single machine's hardware or software, even replacing the server with another, but always keeping the existing number of machines in the computational stack.

Figure 3.2 is an example of horizontal scaling. In that case, it adds more machines to the computational stack, adding two more equal machines to it. Each machine computes part of the task, and the result is the aggregation of the combined effort of all machines in the computational stack.

For horizontal scaling, it is essential to note that there is a software effort required to synchronize different machines to perform the same task, such as MapReduce[8] (DEAN; GHEMAWAT, 2004) or Spark[9] (SHAIKH et al., 2019). So along with all the peripherals, horizontal scaling also adds a managing overhead.

Another crucial detail for horizontal scaling is the intercommunication between the machines. As outlined by Elmirghani et al. (2018), network devices have a vast area to explore in energy efficiency. Because horizontal scaling requires the machine to operate together, communication is an essential part of horizontal scaling that also consumes resources and adds to the solution's overall energy footprint.

Not necessarily horizontal scaling is better than vertical scaling or vice-versa, and the definition of the best solution is debatable according to the scenario it is applied to.

As discussed in Section 3.4, the CPU utilization does not incur the highest energy consumption on a server due to the several other components necessary for the computer to function. Due to the fact that it is not possible to vertically scale beyond a soft upper

[8]https://hadoop.apache.org/
[9]https://spark.apache.org/

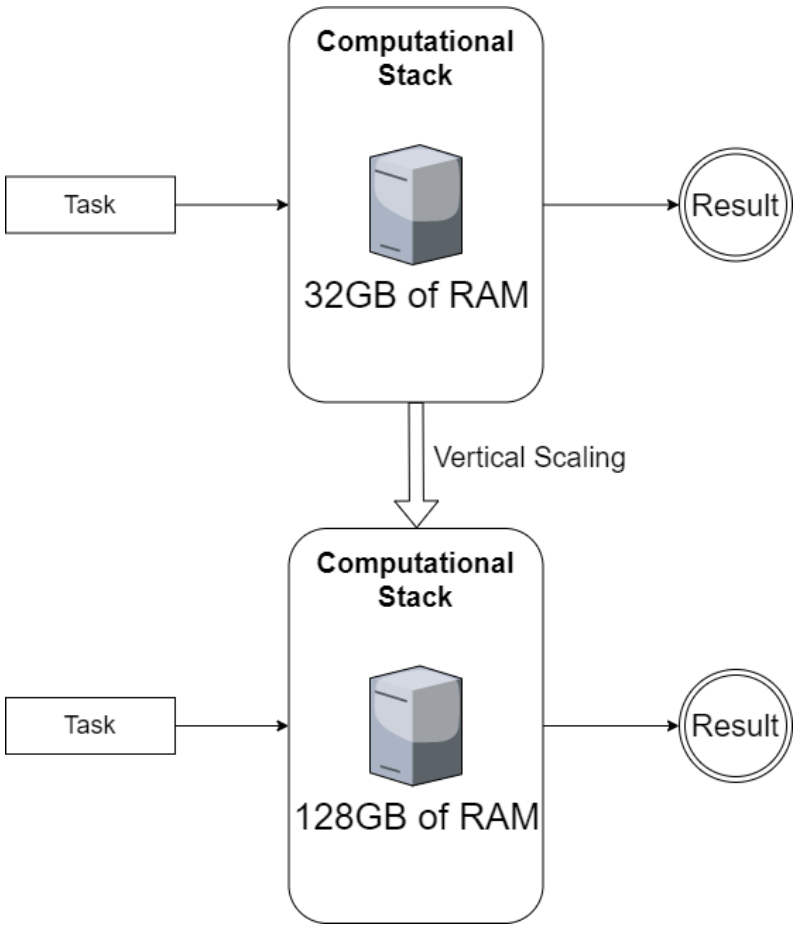

Figure 3.1: An example of vertical scaling. In this example, one machine composes the computational stack. One possible vertical scaling is increasing the amount of installed DRAM. In this case, the amount increased from 32 gigabytes to 128 gigabytes.

limit (due to hardware and technical limitations) and the increased cost to scale vertically, most solutions do scale horizontally, but horizontal scaling tends to spend more resources, having a larger energy footprint.

Due to the facts mentioned above, horizontal scaling has a larger energy footprint than vertical scaling at the commercial and standard levels due to static power generated by computer components, communications, peripherals and external equipment required to maintain the data centers, since each new machine added to the computation stack will have its own set of peripherals and require cooling and controlled power.

The next section presents tools to measure energy consumption.

3.6 Tools for measuring energy consumption

Measuring energy consumption is a critical factor in green computing, and consequently, in green AI. Without measurements, there is no comparison of the actions'

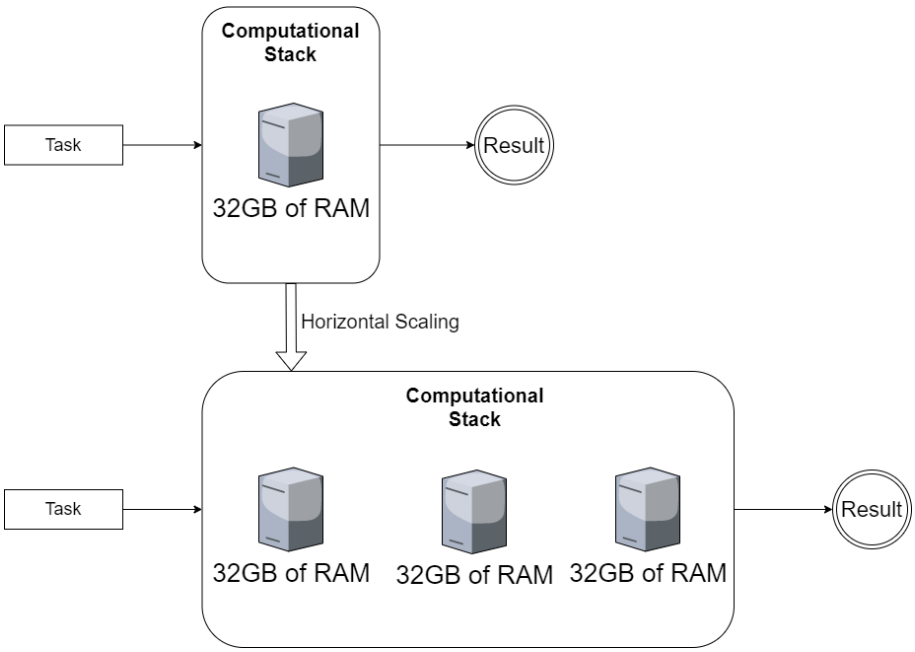

Figure 3.2: An example of horizontal scaling. In this example, the horizontal scaling adds more machines to the computational stack, distributing the work between all computational units.

efficiency, and most of the results are not analytical. There have been numerous successful attempts to measure energy consumption in the past decade, which are described below.

3.6.1 PowerAPI

PowerAPI[10] is a software solution that aims to measure energy consumption at the software level. It provides a Application Programmable Interface (API), measuring at the process level (BOURDON et al., 2013). Since the paper's publication, PowerAPI has evolved in different ways. Currently, it provides a docker interface and expands its capabilities to measure processes in a machine and a swarm of machines and virtual machines.

It consists of two distinct parts. The first part is the sensor that measures one process, group of processes, virtual machine, or machine. The sensor requires a database to save the collected results, used by the energy consumption estimation formula. This formula uses the results collected by the sensor to generate the energy consumed, and then saves the results in an output database that are ready for consumption for analysis or another process.

[10]http://www.powerapi.org/

As outlined by the PowerApi webpage[11], to deploy a sensor to measure the hardware's energy consumption in a Linux-based environment, one must run the following docker command:

```
$docker run --privileged --net=host --name powerapi-sensor \
  --privileged -td -v /sys:/sys \
  -v /var/lib/docker/containers:/var/lib/docker/containers:ro \
  -v /tmp/powerapi-sensor-reporting:/reporting powerapi/hwpc-sensor \
  -n $NAME \
  -r "mongodb" -U "mongodb://ADDR" -D $DB -C $COLLECTION \
  -s "rapl" -o -e RAPL_ENERGY_PKG
```

The $NAME, $DB and $COLLECTION variables must be already in the environment, ADDR is the MongoDB address. $NAME is the name used by the container, $DB is the database name and $COLLECTION is the collection in MongoDB used to store the sensor data.

To deploy the formula (the formula uses the sensor data and saves the consumption in another MongoDB collection), one must run the following commands:

```
$docker run -td --net=host --name powerapi-formula \
  powerapi/rapl-formula \
  -s \
  --input mongodb -u mongodb://ADDR -d \
  $INPUT_DB -c $INPUT_COL \
  --output mongodb -u mongodb://ADDR -d $OUTPUT_DB \
  -c $OUTPUT_COL
```

The second command will deploy the formula docker, that takes data from the input collection, process it, and outputs the consumption to an output database. $INPUT_DB is the input database used by the sensors, $INPUT_COL is the collection used by the input database; both ADDRs are the addresses of the input and output databases. $OUTPUT_DB is the output database, and $OUTPUT_COL is the output collection.

Several works in the literature make use of PowerAPI to provide energy consumption measurements such as: (NOUREDDINE et al., 2012a; NOUREDDINE et al., 2012b; NOUREDDINE; ROUVOY; SEINTURIER, 2013; NOUREDDINE; ROUVOY; SEINTURIER, 2014; NOUREDDINE; ROUVOY; SEINTURIER, 2015; COLMANT et al., 2015; COLMANT et al., 2017; COLMANT et al., 2018) to name a few.

Using PowerAPI to measure energy consumption has several advantages, such as

[11] http://powerapi.org/

measuring multiple locations at once, measuring separate processes, virtual machines, and even estimates the consumption of the network traffic.

The main downside is that it requires a moderate amount of setup and tinkering to make it work. Also, not all features are available for all hardware options (BOURDON et al., 2013).

3.6.2 PowerTOP

PowerTOP is a tool provided by Intel to measure which processes utilize the CPU the most, allowing the monitoring of which processes wake the CPU from an idle state, thus allowing the monitoring of high energy demanding processes (ACCARDI; YATES, 2014).

PowerTOP also provides a set of tuning tools directed towards energy saving, and is provided open source as part of most Linux' standard distribution packages.

To have PowerTOP collecting energy usage and saving it as a comma-separated values file report (CSV), one should run the following command:

```
$powertop --csv=csvfile.csv
```

This command starts monitoring all processes in the machine and produces a report of the energy consumption measured.

It is important to note that powertop requires specific Intel processor instructions and may not produce energy results on all hardware.

There are some works that use PowerTOP, such as (MARTÍN; LAVESSON; GRAHN, 2015), and it is used frequently by Linux's users to verify which processes are consuming resources from the CPU.

The main advantage of using PowerTOP is that it is a tool developed by Intel and is in constant development. Also, because it does not require complex setups and runs with relative ease when the hardware is right, producing reports saved on file automatically is one of the most accessible measurement solutions available.

The main downside is its simplicity, as it requires custom made solutions to measure a multitude of machines, unlike PowerAPI.

3.6.3 Running Average Power Limit (RAPL)

Running Average Power Limit (RAPL) is a driver provided by Intel that allows the measurement of energy consumption read by the chip's meter (a hardware feature that estimates energy consumption at the hardware level). With this driver, it is possible

to measure energy consumption entirely by software. Initially, the tool could measure the energy consumption of DRAM in hardware chips. It provided a set of tools to measure and limit those chips' power consumption, essentially saving energy in the process (DAVID et al., 2010). After ten years of development, the tool acquired new capabilities, being able to measure energy from the processor, core, and uncore components (INTEL, 2014). RAPL is part of the Linux' kernel and requires specific Intel processors to function, specifically, processors built after 2011 codename Sandy Bridge, though not all processors fully support this feature (though most if not all of them do have at least one or two features). For a complete list please refer to Intel's website, as it changes as new processors are developed[12].

According to Desrochers, Paradis and Weaver (2020), there are three ways to read RAPL measurements:

- Read the files in the Linux's directory $/sys/class/powercap/intelrapl/intelrapl : 0$.
- Use the perf interface[13].
- Access MSRs under $/dev/msr$ (MOTOROLA, 2020).

The main advantage of using RAPL is that RAPL is a Linux Kernel driver that reads from a real energy estimator inside the CPU and it already comes bundled with any Linux Kernel version 3.13 or above. Also, it is straightforward to acquire energy measurements using RAPL (read-only files).

The main drawback of RAPL is the impossibility to separate individual processes when measuring. The measurement is always from the entire CPU.

Numerous works make use of RAPL, such as: (DESROCHERS; PARADIS; WEAVER, 2016; PHUNG; LEE; ZOMAYA, 2018; THORAT; INAMDAR, 2018; VENKATESH; KANDALLA; PANDA, 2013) and others.

3.6.4 Hardware Solutions

There are many hardware solutions aimed at measuring energy consumption such as (FLINN; SATYANARAYANAN, 1999; ANDERSON et al., 1997; ZHANG et al., 1997; MONSOON, 2020). They work by measuring the current that flows towards the pieces of equipment. Since current is directly proportional to energy consumed, applying the power formula $P = V \times I$, it is possible to obtain the energy measurement.

[12] http://www.intel.com
[13] https://perf.wiki.kernel.org/index.php/Main_Page

Though in more recent works, most studies focus on using software solutions because over the years, there were works that proved that software solutions measurements are close to the actual energy consumption (PANIEGO et al., 2018).

3.7 Concluding Remarks

This chapter gives the reader a glimpse of the vast body of knowledge of green computing, and in a broader sense, what is green as a whole. It touched green AI aspects and how they relate to data streams, thus bringing attention to static power and machine consumption. Also, it brings the reader insight into the overall machine's consumption and how energy consumption increases as processes grow and require more resources to operate. Finally, it shows some tools to measure energy consumption from a machine. The next chapter presents the related works that lie at the intersection of data stream mining and green computing to the reader.

Chapter 4

Related Works

As separate areas, both data stream mining and energy measurement have many related works, as seen in the works of (XU; WANG, 2017; BIFET; GAVALDÁ, 2007; BIFET et al., 2009; BIFET; HOLMES; PFAHRINGER, 2010; BIFET; KIRKBY, 2009), to name a few. On the topic of energy measurement and energy awareness, the topic is too vast, ranging from the early discovery of electricity, being as broad as areas that use equipment that relies on electricity (DAVIS, 1901; NAKANO, 1889). Software energy measurement has picked the interest of researchers for a long time, but it gained strength roughly at the start of this millennium, and we can cite (DAVID et al., 2010; FLINN; SATYANARAYANAN, 1999; ACCARDI; YATES, 2014), just as few examples among the many works.

Though vastly explored areas, the intersection between the two remains roughly unexplored. Most notably, we have the work of Martín (2020), where the author has explored many facets of energy efficiency on various machine learning techniques, one of those being data stream mining, even producing two novelties: stream mining method called Green Accelerated Hoeffding Tree and Hoeffding tree with $n_{\min}$ adaptation.

In the aforementioned work, n_{min} adaptation for Hoeffding Trees sets a minimum threshold nmin defined by the user. This parameter is used to define if the calculations to determine if a node should be split will run or not. If not enough instances are in the node, the entire calculation is skipped, saving energy and resources. Green Accelerated Hoeffding Trees are Hoeffding Trees that, besides the available metrics, also contain an energy budget. This energy budget is used to determine how many nodes can remain active. The nodes that are deactivated are the nodes with less impact on the overall performance of the tree.

Having published many papers related to the topic, we can mention at least five that are very similar to our proposed work, most of them contained within the PhD thesis

of Dr. Martin (MARTíN, 2020): (GARCIA-MARTIN; LAVESSON; GRAHN, 2017), a study case of energy in very fast decision trees, (GARCíA-MARTíN et al., 2020), an $n_{\min}$ adaptation for very fast decision trees, (GARCíA-MARTíN; BIFET; LAVESSON, 2020a) where the authors model the energy footprint of Hoeffding tree ensembles, and (GARCíA-MARTíN; BIFET; LAVESSON, 2020b) a type of very fast decision tree that is more energy efficient than very fast decision trees and has almost the same accuracy, and (MARTÍN; LAVESSON; GRAHN, 2015), which most closely resembles our work, makes use of PowerTOP to measure different scenarios concerning data stream models.

Though the Dr. Martin's work ultimately leads to two novelties in terms of energy efficiency, some of the work strongly relates to our work, specifically on the topic of measuring energy. Throughout the many works that compound the thesis, Dr. Martin used many approaches to measuring consumption. The real consumption measurements, though, were done through either PowerAPI (BOURDON et al., 2013) or PowerTOP (ACCARDI; YATES, 2014).

Since the work of (MARTíN, 2020) culminated in developing two novelties in energy efficiency it still left a gap to explore: energy measurement in data stream mining. The work Dr. Martin did uses external tools to measure the energy consumption of data stream models.

4.1 Strands of the related works

The related works in the area that intersects both data stream mining and green ICT are divided into two categories: it either aims to study the energy consumption of the areas of the code, such as (GARCÍA-MARTÍN et al., 2019a), (GARCÍA-MARTÍN et al., 2019b), (GARCIA-MARTIN; LAVESSON; GRAHN, 2017), and (MARTíN; BIFET; LAVESSON, 2020), or it aims to improve a particular aspect and optimize the energy consumption, such as (GARCíA-MARTíN; BIFET; LAVESSON, 2020b) that proposes a new model and (GARCíA-MARTíN et al., 2020) that presents an adaptation for the VFDT.

Both cases excel in their areas, particularly in presenting new methods that are energy-aware and have accuracies comparable to their counterpart methods.

4.1.1 Energy Measurement of data stream mining

In the works where the main goal was to measure the energy consumption, the researcher focused on measuring the energy consumption of the data stream mining process

and disregarded the static power consumption of the hardware that ran the experiment. In doing so, the researcher was able to narrow down and pinpoint in the code where were the energy consumption hotspots of the studied models. Furthermore, knowing the parts of the code that consumed energy, the researcher was able to compare the efficiency of each section of the code in terms of energy efficiency versus accuracy gain. Those discoveries ultimately led to the novelties in optimizing the existing models.

4.1.2 Optimizations and new models

In the other works, where the goal was to utilize the knowledge gained from the energy consumption through the code, the researcher was able to optimize the efficiency of the data stream model by limiting at one time limiting the number of operations that were too expensive and did not provide gains (GARCíA-MARTíN; BIFET; LAVESSON, 2020b), and on the other one, directed the usage of resources to operations that provided the most gains, while also limiting the expensive operations that were less useful for the accuracy.

4.2 Gaps of the knowledge

Though both strands are essential and valid, not to mention one leads to the other, they leave a gap in the area of knowledge that we can explore in this work.

The researchers used a software solution running alongside the processes during the measurement-related works. Although that is a valid form of measuring energy consumption, it is by no means trivial and available to the general public. In addition, the software used to measure consumption requires specific knowledge and proficiency. Also, when modeling the energy consumption of a specific data stream mining model, the researcher abdicated the other models, meaning that the mathematical model will work for only a portion of the cases, not all cases.

The works that are the novelties, with a new proposed model and improvements, do not solve the existing models' pain points.

In both cases, the researcher provides an excellent alternative and means to calculate the energy consumption of a data stream mining model but does not solve the following issues that are tackled in this work.

- How does one measure the energy consumption of any data stream model and configuration during its development phase?

- How does one lessen the difficulty in measuring the energy consumption of data stream mining models?
- How to provide a tool available for the general audience of data stream mining?

4.3 Concluding Remarks

In light of the questions left behind by the related works, this work has the following highlights: proposes a plugin that integrates the energy consumption of the model directly in the exploratory tool so that the scientist can be aware of the chosen model's energy consumption, enabling a better judgment that is not solely based on accuracy but also energy efficiency.

An integrated plugin in the modeling tool helps alleviate the pain of measuring the energy consumption because there is no need to purchase specific hardware or install and control additional software. Also, an integrated plugin helps in being a more available tool for the general audience because it reduces the entry barrier to start measuring.

In addition to the above reasons, this work does not make any assumptions about the configuration being measured, meaning the researcher can choose any desired configuration and will have a result that can be used as a comparison metric.

In this chapter we presented the state of the art in the area of energy consumption in data stream mining. The next chapter presents the user with the scientific method utilized.

Chapter 5

Scientific Method

The type of research used in this work is mixed, as it contains elements of qualitative research, and quantitative research.

This study is classified as qualitative since it provides a plugin to lessen the pain of measuring.

This study is also quantitative when comparing the different configurations of the data stream models. It uses metrics in order to establish a comparison between said models. That is, it makes use of data and quantifiable numbers. According to (GIL, 2007), quantitative research uses mathematical models and parameters that can be numerically quantified.

In addition, it uses experimentation. According to (RUDIO, 1980), the experimentation method is necessary for the experiment to achieve reproducibility and produce the same results. It is essential to mention that the quantitative results obtained can only compare measurements done using the same hardware and software running alongside them. Therefore, the results obtained are particular to that configuration and vary even when using hardware and software with the exact specifications.

To find a suitable hardware for the experimentation, the researcher opted for a search in a primary source to find a suitable hardware, mainly resorting to google for the search. The keywords used were "smart plug," "intelligent plug," and "energy measurement plug."

When finding the software solution, the method utilized was the literature review using secondary data sources. Following both the sources of the bibliography present on the data stream mining and searching in the databases with the keywords "energy AND measurement AND software."

This study is also classified as a comparative study because it compares the energy consumption of different configurations of stream mining models. According to (COL-

LIER, 1993), comparative studies seek the relationship between elements and consider their differences and similarities to establish a relationship between them. Therefore, particularly in the case of this work, it compares different configurations and provides information for an informed decision.

5.1 Overview

One of the obstacles preventing researchers from measuring the consumption of their data stream mining models during its initial phases is the difficulty of having a measurement setup. While measuring itself is not a complex task, and most individuals can obtain a piece of equipment or software to measure energy consumption, many barriers discourage this practice. Also, the lack of a bigger picture regarding the impact the model will have in the future does not compel the researcher to procure a more energy-efficient model.

One can name the following: a laborious measurement setup that applies to most measuring devices, either by hardware or software. If opting for a hardware solution, measuring devices will incur a monetary cost to the researcher. Also, the lack of knowledge to measure: To properly use the selected solution, one must know how to operate it.

For example, as an exercise, one can imagine a researcher with no electricity measurement background. He will probably not know what equipment to select, or he may not even be aware of the data stream model's energy consumption. Imagining, he discovers that one can measure consumption using a hardware solution. He discovers that by measuring the current passing by the energy wire to the computer, one can determine how much power it is consuming. Now, he has to get a computer power cable, strip one of its wires, connect the probe to the cable, and take notes of the current going through the cable for the duration of the experiment. After that, the researcher calculates the consumption using the acquired data. On the other hand, if he decides to use a software solution, he might not be familiar with the tool, not have the proper equipment and be unfamiliar with how to perform the measurements.

5.2 Premises

There are three premises for not measuring the data stream model's energy consumption during its development stages:

- There is a cultural complacency regarding the topic: Most developers will disregard

the measurement because the metric is not commonly used.

- There are only a few tools to measure consumption, and when one searches for them,
- The tools are too outdated, expensive, or complex to use.
- There are no tools that can be incorporated into the existing modeling software, lessening the pain of measuring energy consumption.

As a proposal, this work proposes creating an energy measurement plugin for one of the most commonly used software MOA (see Section 2.6.1) to develop a data stream model that would help alleviate the measurement gap in this area.

5.3 Plugin development method

In order to properly create and validate the plugin, those are the steps that must be followed:

- Validated that it is possible to measure the energy consumption of a data stream model during its development phase.
- Having confirmed the previous statement, verified how to read the consumption flags, either by hardware or software tools.
- Verified how to control the learning cycle of a data stream model in MOA.
- By having complete control of the learning cycle, decided which phases it was desirable to measure the model.
- Inserted the measurement function on each desired phase and relayed that information to the software.

5.3.1 Inclusion and exclusion criteria of the measurement hardware

Many solutions can measure the energy consumption of the energy that passes through a wall plug, so the researcher created some criteria to select the equipment.

- The device must measure the energy used by the device within a certain period.
- It is not necessary to modify the hardware to measure.

- It does not measure the energy manually, meaning it is possible to acquire the values automatically.
- There is no need to annotate the values and transfer them manually to the computer.

After some consideration, the researcher opted for the TP-LINK HS110[1] because it was accessible, was economically viable for the project, provides the information via API[2] calls in the network , and does not require modification of the hardware to connect the device to the computer.

After the selection, the researcher purchased the device. Sequentially, performing the initial configuration as instructed by the instructions manual in the package, the device is ready to connect to the available app. The researcher did not use the app because it does not provide the necessary integrations but used a third-party library[3] that accesses the device and provides the details on the various metrics and functionalities of the device. One of the available data is the energy consumed by the device. This information is then used to get the energy spent.

5.3.2 Inclusion and exclusion criteria of the measurement software

Even though hardware provides an accurate solution to measure energy consumption, it is a solution that has potentially less coverage than software. However, a software solution is only limited by the capability of the computer's hardware where it is running. So, as long as the computer where the model is being researched supports measuring the energy consumption via software, the measurements are available without needing additional equipment.

For the software, the selection criteria included, just like for selecting the hardware, one that enabled controlling the acquisition of the measurements by a command, like an API or directly controlling it. Such requisite is essential because by controlling when the measurement is performed and providing the timestamp, the measurements are not compromised due to temporal differences. Also, the software measurement must still be functional, meaning that it does not have requirements that include other software or hardware that has already reached its end of life.

One verified three possible software solutions found in the literature revision that meet the earlier criteria: PowerAPI (see Section 3.6.1), PowerTOP (see Section 3.6.2), and RAPL (see Section 3.6.3).

[1] https://www.tp-link.com/us/home-networking/smart-plug/hs110/
[2] API - Application Programming Interface. It is a way to interact and communicate with the systems
[3] available at https://github.com/jkbenaim/hs100

In order to select one of the tools, it is essential to test them for their correctness, ease of use, and the possibility of incorporating them into the plugin. So the selection criteria, in this case, is subjective to the researcher that arbitrarily decided that RAPL was the easiest one to use and incorporate in the plugin, as all solutions provided with very similar, if not equal, measurements.

To compare the measurements, the researcher starts the software solutions simultaneously, directing their energy consumption output to separate files. In order to provide a process that consumes energy on the computer, the researcher used a CPU stressor, stress-ng[4], for some time. Then, one compares the graphs generated by the consumption. It is crucial to have the same measurement interval from one solution to another and output the timestamps so that the results can be normalized and adequately compared.

5.3.3 Comparison of the hardware tool against the software tool

This method is related to the first experiment described in this work (see Section 6.2 and Section 7.1).

Having selected a hardware tool and a software tool, and in order to provide a more generalizable solution to measure, the researcher compares the software solution against the hardware solution.

To perform this comparison, the following steps are taken:

- Prepare the experiment that will consume energy from the computer. This step can use any tool or process, but the researcher opted to train a model in MOA.
- Start the two solutions, ensuring that both outputs contain the energy consumed and the timestamp.
- Compare the results obtained, ensuring that all variables are accounted for, such as time and power.
  - It is expected that the results to differ significantly because the software tool does not consider the static power. In contrast, the hardware solution will provide a complete hardware measurement.
- Apply the correlation between the results to verify if they are comparable.

Suppose the correlation between the measurements taken by the hardware tool and the software tool is high (between 0.5 and 1.0). In that case, the software solution is

[4]https://wiki.ubuntu.com/Kernel/Reference/stress-ng

possibly an excellent alternative to the hardware tool. The higher the correlation between the measurements, the better. It is important to note that since the measurements and comparisons are related to each specific hardware, we cannot compare the software measurements between different hardware. Furthermore, even using the hardware tool that provides the actual measurement of the system, the actual measurement is not comparable to measurements performed on other hardware due to their specific characteristics such as static power, efficiency, and computational power.

When researching, one must commit to choices. One of the choices made in this work is to use the total measurement of the hardware versus narrowing it down to the data stream mining process. The researcher opted for the total measurement once one reckons that this is the manner that most systems work when at scale. However, this metric may not be ideal for comparing the models. This choice was made when assuming that those models would be allocated in dedicated computational units and is necessary to measure that model and all the auxiliary processes running along with it. Also, it is assumed that when running in shared hardware, where the individual metric makes more sense, it does not have a significant energy impact.

The researcher took the following measures to mitigate the interferences in the final results:

- There is no GUI (Commandline only)
- All non-essential running processes were stopped
- Unplugged any non-essential hardware
  - Mouse
  - Keyboard
- No internet connectivity

5.3.4 Development of a plugin for MOA

With an easy and available tool, the researcher who develops data stream models will have an easier comparison of the energy efficiency of two learning configurations. Because of this, this research proposes to develop a plugin as a tool to help compare the energy efficiency of data streaming models.

Plugin development method:

- Develop the routine to obtain the energy consumption of the processor. There are at least two methods of collecting the data that are usually used for the development of a routine:
  - Battery charge data (only applies to battery-operated notebooks)
  - Use of consumption registers (this is a more generic solution and does not need a battery). For this research, the researcher used consumption registers, as it was what would be relevant to the hardware used in the study.
  - After completing the function, the researcher will have the processor's energy consumption at a given moment.
- With the data obtained in the measurement process, this data must be made available to MOA. This is done by adding a new key to the summary passed to MOA.
- The measurement frequency was defined based on the one already used by the modeling software not to burden the system and, therefore, the machine's energy expenditure. At each determined number of training cycles of the model, a call was made to the energy measurement function, adding the result to the summary of the execution of the experiment.
  - As a result of this process, in addition to the information already made available by the modeling software, information on the energy expenditure of training this model will be available to the user.
  - In addition to that, at each moment when the energy is measured, the user has the instantaneous consumption of the model.

5.3.5 Comparison of the energy consumption of different data stream model configurations

After coding the plugin, the researcher wants to compare the energy consumption of the data stream models when there is a change in one variable to quantify the variation, whether it increases, decreases, or does not affect the energy consumption.

The measurements are performed in the following manner:

- Open the MOA with the plugin loaded.
- Select the experiment.
- Modify the options in MOA according to the manual as desired by the researcher.

- Run the experiment, waiting for its completion.
- Save the results.
  - One of the columns on the summary of the experiment is the model's energy consumption.
- For the next experiment, change one parameter to verify the impact of that particular parameter.

5.3.6 Measurements comparison

To compare the measurements, one must establish a baseline of comparison. For example, should we normalize the energy cost by its power(energy spent per second)? Both measurements are available to the plugin, but one should not use the power, as it would not measure the energy spent by the model's training process. However, instead, one would be measuring the power of the hardware that runs the experiment. Furthermore, in general, all models would have very similar values. This is because the operations performed from one experiment to the other are similar, and the hardware would be running at almost the same capacity. Therefore, it would yield similar results for power.

By comparing the total energy spent by the experiment, one can have a better overview of the efficiency of each model. Some metrics would be possible, for example, consumption for each instance given to the model. That would be an interesting metric, but since this work used the same amount of inputs for all experiments, the results would be the same. By comparing the total energy spent by the experiment, one can have a better overview of the efficiency of each model.

5.3.7 Concluding remarks

This chapter presented the scientific method used in this work. It starts by defining the types of research used throughout the process and how this piece fits in each described category. Each step of this work is then explained, with detailed instructions on performing them. In the next chapter, the reader is then presented with the setup for each experiment and how the plugin is developed.

Chapter 6

Experimental Setup and Plugin Development

The previous sections of this work introduced the concepts of data stream mining and green IT. As we introduced data stream mining concepts, such as concept drift, different validation methods, and how it copes with vast amounts of data, we also pointed some flaws, specifically when it concerns resource consumption. Regarding green IT, the reader went through some energy consumption concepts, such as static power, and methods to measure energy consumption. This work focuses on the lack of easily available energy consumption measurement tools during the development period of data stream mining models.

In this chapter, Section 6.1 describes the plugin developed in this work, and Sections 6.2 through 6.6 describe the experiments to validate the plugin, which is then followed by the experimental setup implemented to evaluate both the plugin and machine learning algorithms. Finally, on Section 6.7, we describe the instantiated plugin for MOA to measure energy consumption. Section 6.8 describes how each research question is answered.

6.1 A Plugin for Assessing Energy Consumption in Data Stream Mining

The plugin is a part of the method to measure the energy consumption of data stream mining models, so the basic idea, shown in Figure 6.1, is controlling the models' training and testing phases. During each of the phases, take the energy measurements; since it iterates many times between these cycles, there are many aggregation patterns for the values.

In this work, the researcher decided to use the same rate of sampling used by the

software MOA, not separating each step for two particular reasons:

- When measuring, the measurement itself incurs some load to the CPU. Also, each iteration is composed of three steps:
  - Prediction - happens at every iteration.
  - Training - happens at every iteration.
  - Evaluation - happens at intervals. Because of the other operations during the evaluation, it might incur an undesirable load on the machine.
- The update frequency of the energy consumption on CPUs happens at regular intervals but not at every operation done by the CPU. When acquiring the consumption too often, the researcher will not obtain any new data, only acquiring the past measurements. Thus, it is desirable to measure at a lower frequency.

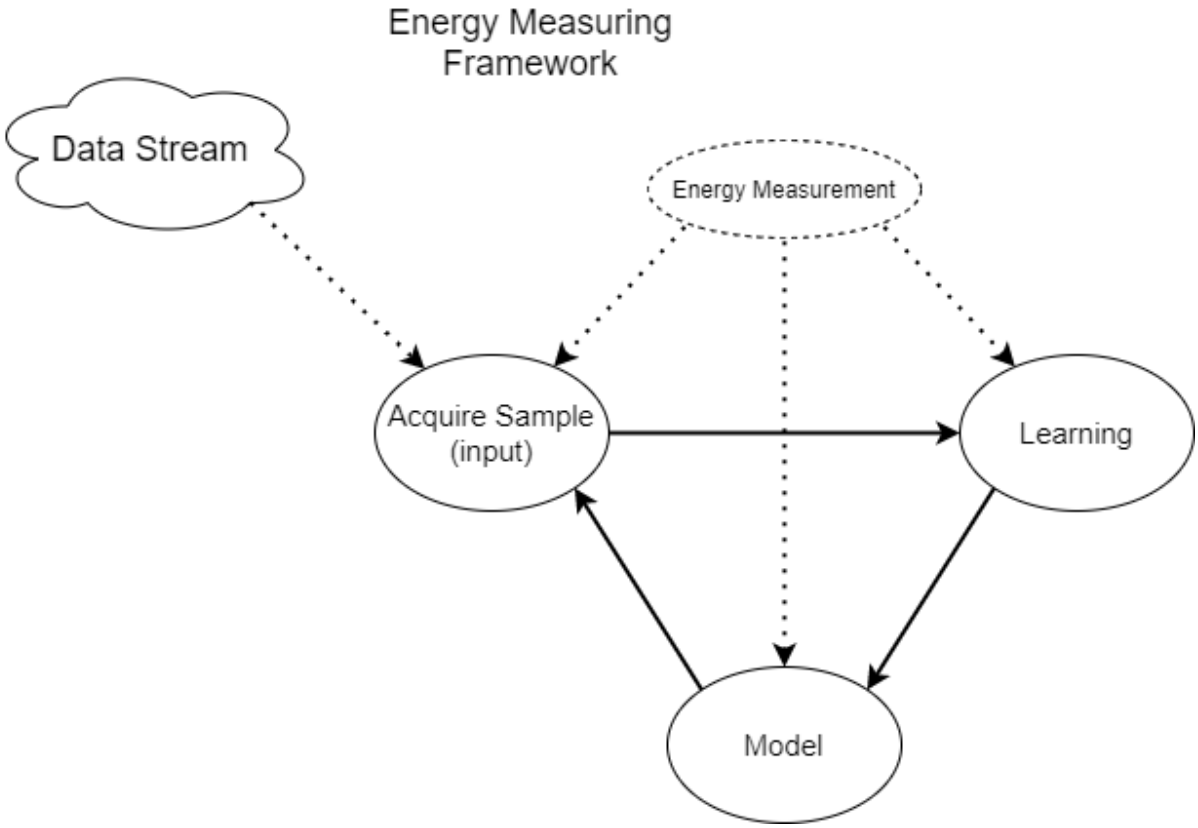

Figure 6.1: The core of the framework, measuring energy consumption at each step of the data stream model. We will have how much the model spends to acquire samples, how much it uses during its training phase and how much it uses during its prediction phase.

For each of the stages, there are different forms of measuring energy consumption. The framework controls the flow of the data stream model to start the energy measurement

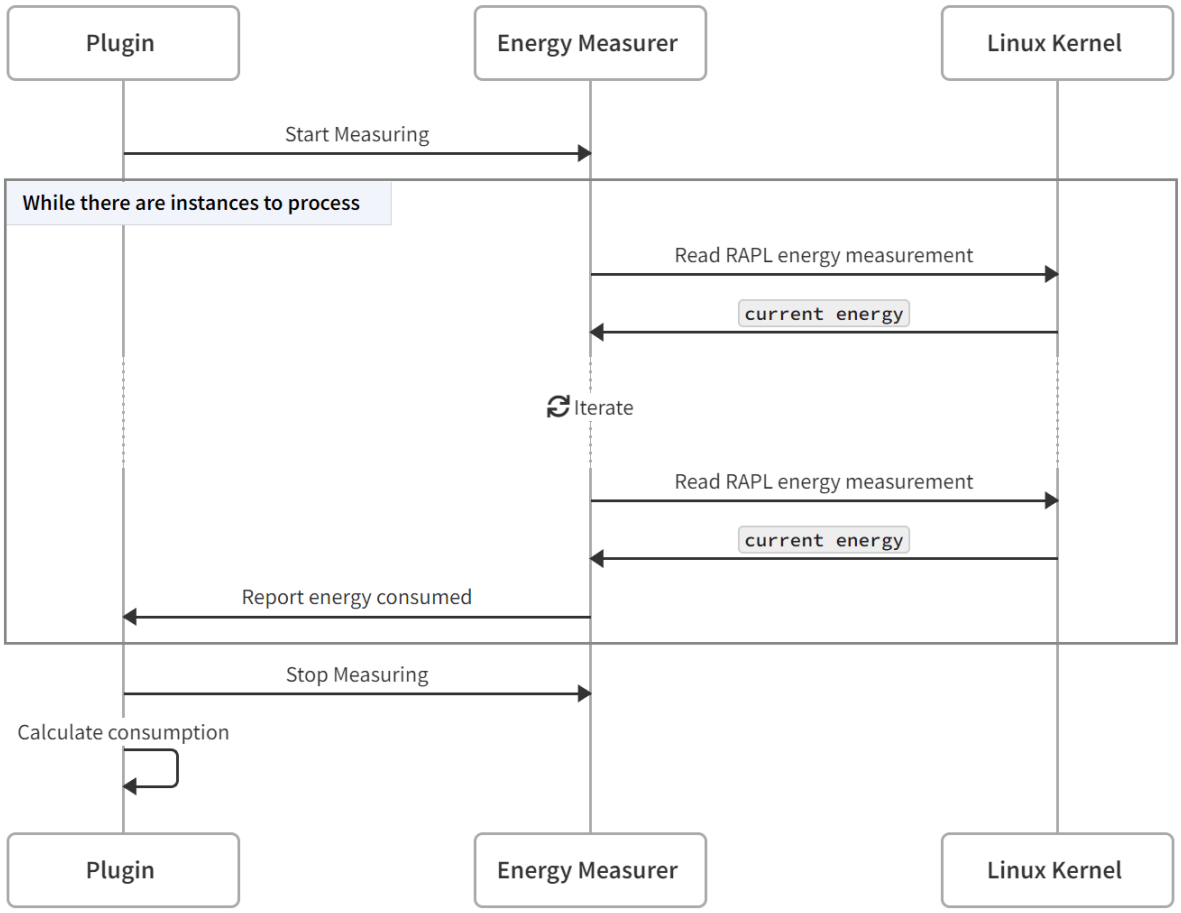

Figure 6.2: Measuring energy consumption using RAPL.

right before it begins processing the samples. At each cycle, the measurement is taken and presented in real time to the user. At the end of the process, a graph showing the instantaneous measurement for each cycle is presented to the user.

The measurement section is seen on Figure 6.2, where the plugin requests measurements from the Linux Kernel, receiving the response of how much energy has been spent during that time.

The code of the plugin can be found at https://github.com/ericonuki/moa-bringing-awareness-green-ict.

6.2 Experimental setup

Four experiments were conducted for this research, which are divided into two phases. The first phase aims at validating software solutions to measure energy consumption and is composed of one experiment. The other phase is to validate the necessity of

the energy measurement plugin (see Section 6.7) and is composed of three experiments.

6.2.1 Overview of the experiments

The first experiment measures how a software solution fares when compared to a hardware solution. In order to do that, a computational intensive task is ran on the machine, while two different methods of measuring energy consumption are plugged onto the computer. One of the tools used is a hardware tool that can measure the energy consumption directly from the outlet (see Section 3.6.4), while the other is a measuring tool built-in the CPU of the computer, that calculates the energy used by the system (see Section 3.6.3).

The main objective of the first experiment is to validate if the built-in energy measurement tool (RAPL) and other software solutions can be used to have a rough estimate or correlation of the computer energy consumption within a certain margin of error compared to hardware solutions.

The second phase, for all three experiments, they consist of producing timed data stream mining experiments, using different data stream mining models, validations and having or not a concept drift. The rationale is to compare how much energy each model consumes under the same scenario, trying to simulate a real-world application, comparing what are the effects of each change.

6.3 Experiment Phase One: Validation of Software Energy Measurements as Viable alternatives to Hardware Measurements.

This experiment Validates the use of RAPL or PowerTOP as tools to measure the energy consumption of Data Stream Mining models.

6.3.1 Experiment One Setup

To validate software solutions to measure overall consumption, we will use specific hardware with the following requirements:

- The equipment's CPU must be Intel Sandy Bridge (2011) or above, as RAPL was developed by INTEL and integrated with some of their chips starting in Sandy Bridge, 2011.

- The equipment must be compatible with RAPL technology, because not all chips contain all functionalities from RAPL.
- The Linux kernel must be 5.4 or superior, as older Kernels might have compatibility issues, and the RAPL interface is surely present from this Kernel onwards.

It is also essential to note that the equipment is exclusive for the model, having no other tasks to perform, preferably running the command line only.

The hardware in which the Data Stream Model runs on requires some specifications properly make use of the measurement tool available in some processors.

Steps for the experiment:

- With the equipment not in use, make sure there are no processes running in the background (daemons, etc);
- Start both measuring tools, taking notes of the timestamp;
- Start the CPU stressor, leaving it on for roughly 15 minutes;
- Stop the CPU stressor;
- Stop the measuring tools.

During the experiment, it is mandatory to not use the computer, as it influences the measurements.

6.3.2 Experiment Phase One Hypotheses

- When compared to the software solution, the energy consumption result given by the hardware solution are equal or highly similar.
- If different, the results between the hardware and software are correlated.

6.4 Experiment Phase Two: Measurement of the Energy Footprint of Well Known Data Stream Mining Models, and evaluators with and without drift

The second experiment will measure how each data stream mining model behaves in terms of energy consumption and will serve as the basis for developing the plugin. The second phase experiment can only proceed if the first phase proves that software alternatives are reliable to measure the machine's consumption.

6.4.1 Experiment Two Setup

This experiment uses the same setup from the previous experiment (see Section 6.3). Start recording the consumption using the plugin constructed over RAPL. It starts a script that performs a predetermined training and prediction for each data stream model. Record the instantaneous and total energy consumption of the experiment, along with the model's accuracy. The other metrics available to MOA are also available during the experiment output.

6.4.2 Experiment classifiers, drift detectors, and validation scenarios

For the experiments, we use the following classifiers:

- Naive Bayes (Section 2.4.1)
- kNN (Section 2.4.2)
- VFDT (Section 2.4.3)
- Oza Bagging (Section 2.4.4)
- ARF (Section 2.4.5)

We also use the following evaluators in this work:

- Prequential (Section 2.5.1)
- Interleaved Chunks (Section 2.5.2)

6.4.3 Stream Generator

For the second phase, the generated stream used for the training was: `generators.RandomTreeGenerator` with the following parameters:

- -c 10 → number of classes
- -o 10 → number of nominals
- -u 10 → number of numerics
- -d 10 → max tree depth
- -l 7 → first leaf level

On the experiments where there was a concept drift, the researcher used `ConceptDriftStream` with the following parameters:

- First Stream:
  - `generators.RandomTreeGenerator`
  - -c 10 → number of classes
  - -o 10 → number of nominals
  - -u 10 → number of numerics
  - -d 10 → max tree depth
  - -l 7 → first leaf level
- Second Stream:
  - `generators.RandomTreeGenerator`
  - -o 10 → number of nominals
  - -u 10 → number of numerics
  - -v 3 → number of vals per nominal
  - -d 3 → max tree depth
  - -l 2 → first leaf level
- The drift occurred at instance number 499,999 and lasted for 499,999 instances.
  - -p 499999 → position of the drift
  - -w 499999 → width of the drift

Thus, the full stream command is:

```
-s (ConceptDriftStream
    -s (generators.RandomTreeGenerator
        -c 10
        -o 10
        -u 10
        -v 10
        -d 10
        -l 7)
    -d (generators.RandomTreeGenerator
        -o 10
        -u 10
        -v 3
        -d 3
        -l 2)
    -p 499999
    -w 499999)
```

6.4.4 Online vs Offline training

For all experiments, it is important to note that all training is performed online, and there is no offline training, which means that all data comes from the data streams during the modeling, and there is no available data prior to initiating the training.

6.4.5 Experiment Two Hypotheses

- The energy consumption of different data stream models is different, even when using the same inputs.

6.5 Experiment three Setup

Experiment three used the same setup as experiment two (see Section 6.4.1), which includes the stream generator and classifiers but in this experiment, the researcher used the plugin developed for this work (see Section 6.7). The added benefit is that the data is available without accessing external files.

6.5.1 Experiment Three Hypotheses

- The energy consumption of Interleaved Chunks will be lower than the consumption of Prequential because it processes more instances at a time, thus being more optimized.

6.6 Experiment four Setup

The fourth experiment used the same setup from the third experiment but changed the classifiers, so they contained drift detectors. The following classifiers were used:

- Naive Bayes (Section 2.4.1) with DDM (see Section 2.3.2.1).
- kNN (Section 2.4.2) with EDDM (see Section 2.3.2.2).
- VFDT (Section 2.4.3) with DDM.
- Oza Bagging (Section 2.4.4) with ADWIN (see Section 2.3.2.3).
- ARF (Section 2.4.5) with ADWIN.

6.6.1 Experiment Four Hypotheses

- When there is a concept drift to an easier concept, there will be a reduction in consumption because the model will require fewer resources to learn.

6.7 MOA Plugin

The usual way to extend MOA is by constructing a new class that interact with the core MOA API. In MOA, it is possible to develop new classifiers, data generators, validation tasks, evaluation metrics, and other aspects related to data streams and their mining. In this work, we developed a new set of tasks that are be responsible for the utilization of a known classifier and validation process. The task conducts the original validation process that is already available in MOA, and it complements it with measurements of how much energy such process uses to train and operate under different scenarios, i.e., classifiers and data streams. The prototype is implemented as a plugin written in Java. The final version is available for MOA, specifically in the tasks of "Evaluate Prequential" and "Evaluate Interleaved Chunks". The current GUI of MOA can be seen on Figure 6.3, and the modified summary can be seen on Figure 6.4.

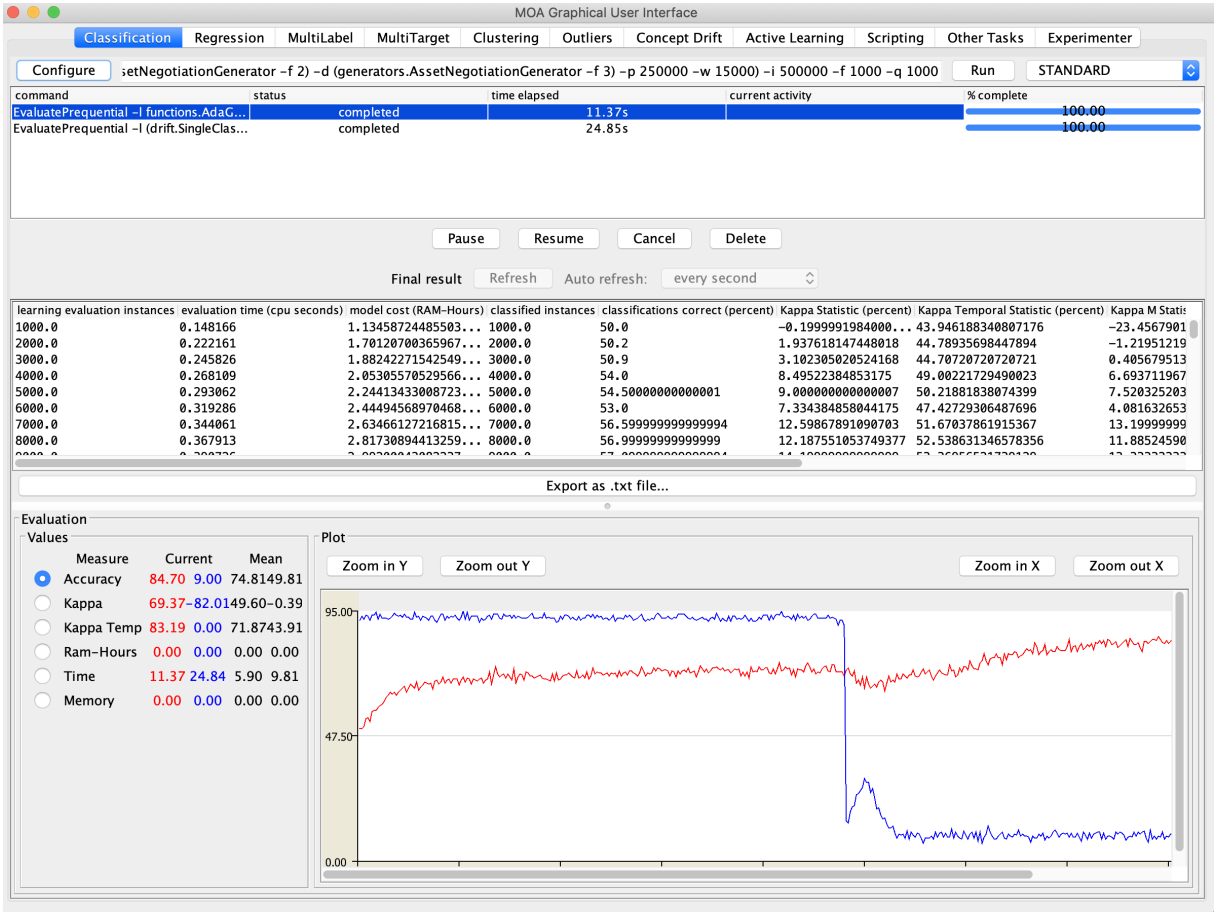

Figure 6.3: Existing Massive Online Analysis (MOA) Graphical User Interface. Screenshot taken from MOA running on a MacOS

The plugin source code is available at https://github.com/ericonuki/moa-bringing-awareness-green-ict.

6.8 Research Questions

Each experiment provides answers to the questions proposed at the beginning of this work.

- How to lessen the pain of measuring the energy consumption of data stream mining models during their development phase?
- What is the impact of the validation protocol in terms of energy consumption?
- What is the impact of drift detectors in energy consumption?

The researcher provided a plugin for MOA that does not require any additional hardware to be installed to answer the first question. Next, the most suitable answer

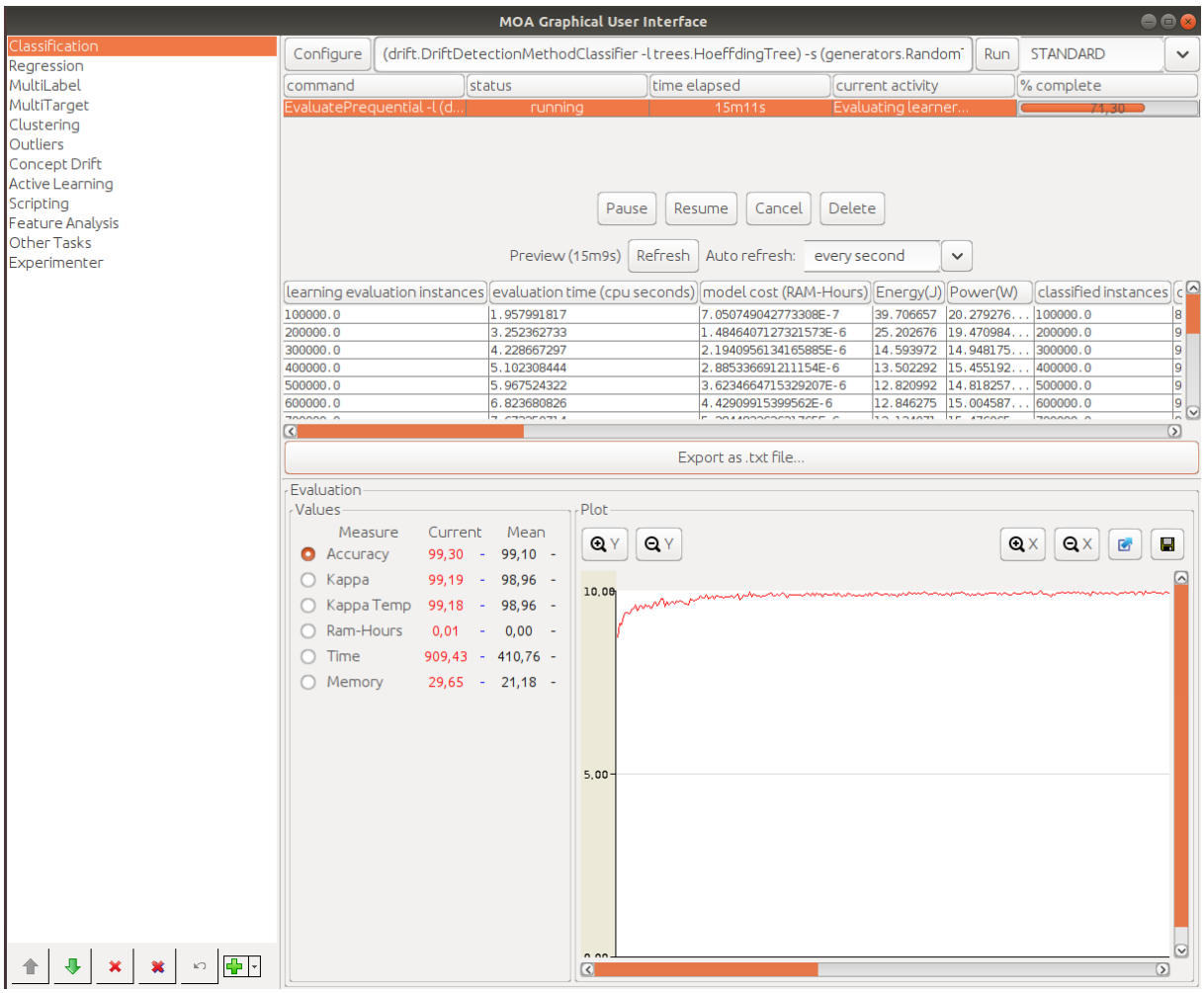

Figure 6.4: Screenshot taken from MOA running with the plugin on Ubuntu Desktop.

to the second question lies in the experiments chapter (see Chapter 7). Sometimes, the desired accuracy may require vast amounts of training to reach, while it may be possible to reach a reasonable solutions utilizing just a fraction of the resources. Other times, the model may have a considerable memory footprint, requiring an extensive effort for predictions. That impacts the performance and has a considerable energy consumption, while a lighter model may have slightly lower accuracy while being much faster and requiring fewer resources for predictions.

For the first experiment, the expected result is finding a consistent battery efficiency factor relating to the software consumption with the measured hardware consumption. Having this result, is it possible to rely on software metrics for the next steps, and it is essential for the development of the plugin, as it will use the same underlying mechanics the tested tools use.

The second experiment expects to compare different data stream mining classifiers, validation scenarios, and drift detectors, with accuracy and energy consumption, to order

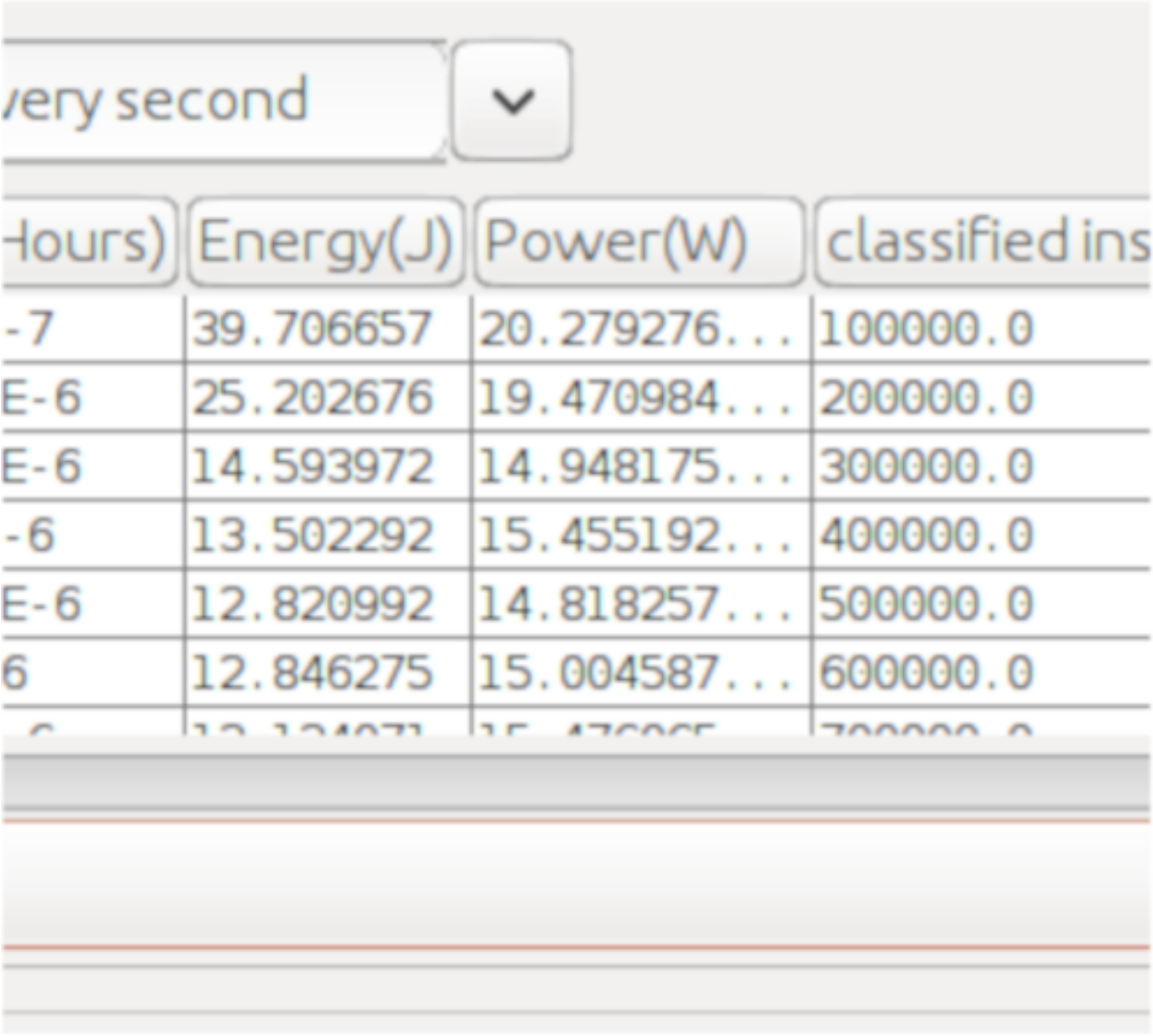

Figure 6.5: Detail of the energy results given by the plugin.

which classifier is more efficient at training and predicting in terms of energy efficiency.

6.9 Concluding remarks

This chapter presented the proposal of this work and the steps to achieve the ultimate goal of bringing energy consumption awareness when tailoring data stream models. To accomplish this goal, first, we have to validate an accessible form to measure energy consumption with confidence. If possible, this method must be accessible due to pricing and hardware setup, and thus, hardware solutions are not preferred, even if they present the best accuracy. Next, we defined control experiments to measure against the chosen methods, using a hardware solution, and the selected solutions. Once accomplished, it is targeted the development of a plugin for displaying energy consumption in MOA.

This validated plugin is the byproduct of the framework, which will help data stream researchers and practitioners to develop and select energy-efficient models in the future. The next chapter presents all the experiments results to the reader, along with some remarks regarding the experiments as a whole.

Chapter 7

Experimentation

This section analyzes results obtained from the experiments conducted to validate a plugin aimed towards measuring the energy consumption in data stream mining. Data stream mining and green computing are new areas of data mining that are gaining momentum in the 21st century. Furthermore, organizations and governments are looking to save energy due to climate change and global warming. Thus, developing energy-efficient models that can give accurate results while maintaining a relatively low energy footprint is a significant breakthrough in the area.

The main research question that guided this study was: how to provide the general public with a means to measure the energy consumption of data stream mining models during their development phase. Thus, the researcher sought to know the impact of different models, validation, and drift existence on energy consumption while measuring it various tools and a plugin developed to measure the energy consumption.

Having the above guidelines will lead to a plugin to measure the energy consumption of data stream mining models that will help evaluate the model's energy footprint during its development.

For all experiments, the following computational setup was used:

- A desktop computer running Ubuntu Desktop 18.04 LTS,
- CPU Intel Core i7-2600 Sandy Bridge,
- 4GB RAM, and
- 250Gb HDD Hard Drive

7.1 Experiment 1 - Comparison of measurements done through a hardware solution and a software solution.

The first experiment was conducted to determine if the energy consumption measurement given by a software solution can be comparable to a measurement given by a hardware solution. Both software and hardware solutions are valid means to measure energy consumption during data stream mining. Thus, in this experiment, hardware and software energy measuring tools were plugged into the computer, and their results were compared simultaneously. The results of the experiments are shown in the Figure 7.2. The experiment showed that although the software solution does not accurately describe the whole consumption of a computer, the results are correlated to the actual consumption measured by the hardware tool.

In this experiment, using the setup mentioned in the introduction of (Chapter 7), the first experiment is set up according to Figure 7.1, i.e., the computer is connected to a wall plug via a watt-meter, which is responsible for measuring the energy consumption. On the computer, a CPU stressor called Stress-ng is installed. Stress-ng is a tool that allows the computer to be stressed by a certain percent of its total capacity, making it ideal for gauging different consumption levels. Therefore, it was decided to stress the CPU by levels from ten percent to one hundred percent, with ten percent step increments, maintaining the stress for 10 minutes for each level. After the computer reached one hundred percent usage, then more cores would be recruited until all four cores were used in the experiment.

Figure 7.2 shows how the computer consumed energy over time during data stream mining. The bars indicate various stress levels in the computer. The blue bars indicate the software energy measurement, while the green bar shows the actual hardware energy consumption measurement.

Clearly, from the figure, it can be concluded that hardware and software assessed energy consumption rates are correlated. Furthermore, when correlation over the results was conducted, it showed a correlation of 0.99972. Thus, the correlation r=0.99972 is very close to one, showing a high degree of correlation between the two measurement tools. Furthermore, the value is positive, meaning that they are directly related.

The results indicate that using the software tools to measure energy consumption during the data mining stream is an alternative approach to using the hardware solution to measure energy consumption during data mining accurately. The experiment's findings are consistent with the other studies, such as (DESROCHERS; PARADIS; WEAVER,

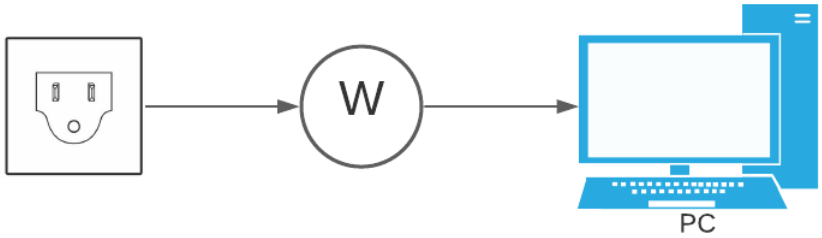

Figure 7.1: Setup for the first experiment. The computer tower is connected to a wattmeter that is connected to the wall outlet.

2016), (PHUNG; LEE; ZOMAYA, 2018) and others, which showed that RAPL is consistent with the actual energy consumption of the computer.

It is essential to note the difference in measurement taken by the hardware solution against the RAPL software measurement. This is because the RAPL only measures the consumption done at the level measured with hardware counters, such as CPU, memory, network traffic, and other inner components of the computers. On the other hand, a hardware device will capture all consumption, including the static power consumed by the computer, even in an idle state. For example, the hard drive will keep spinning even if

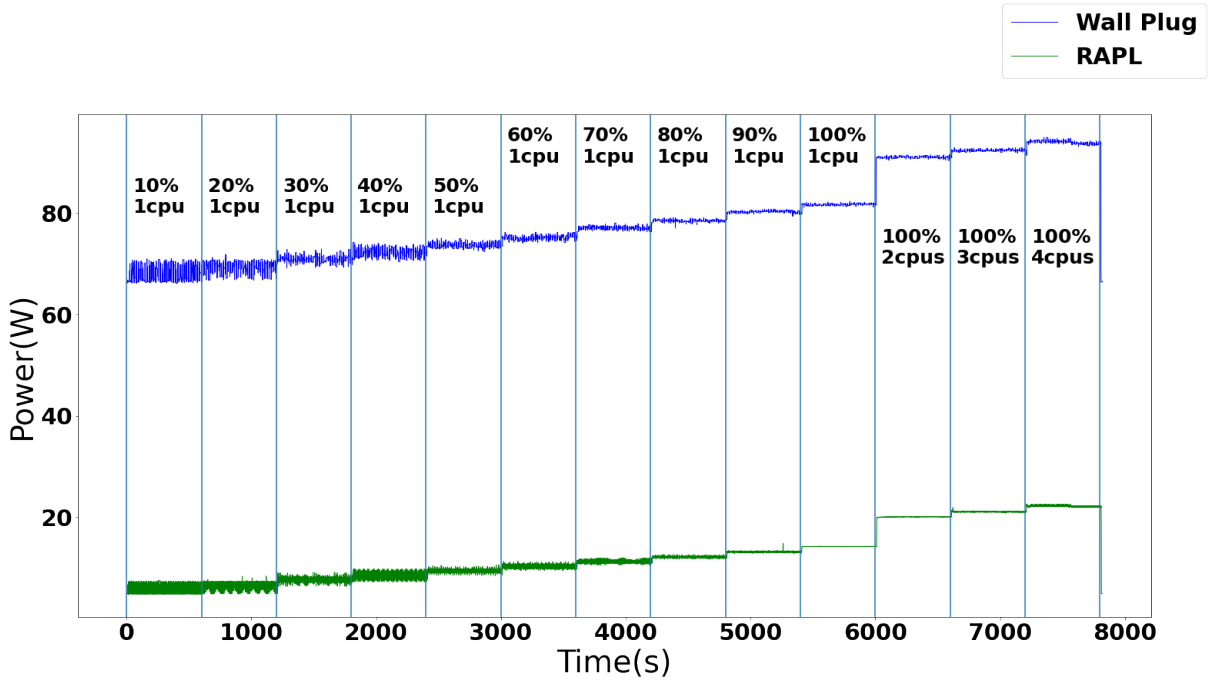

Figure 7.2: Experiment 1's results. The graph shows the energy consumption measured by a hardware solution(in blue) and a software solution(in green). In this experiment, the software solution used was RAPL.

no data is accessed. All the hardware inside a computer will consume energy, measured by the hardware solution. Though different, they are proportional and can be an appropriate comparison measurement for the model developer.

7.2 Experiment 2 - Comparison of different data stream classifiers energy consumption

The second experiment that was conducted aimed at verifying if different classifiers yield different energy consumption rates and if the results could be generalized. These measurements were conducted using the software solution `PERF` with `RAPL` measurements. The software was employed in the second experiment helped in measuring if the classification approaches affected energy consumption. For each experiment, we varied either the evaluator or the classifier, so their performance could be measured and compared accordingly.

The plots shown next represent the power consumption over time for each experiment. While they seem different, they follow a trend of high and low energy consumption. At the peak of energy consumption, the whole computer has more stress, such as when accessing or saving to memory, when the disk is being used. Those situations vary from model to model, but in essence, high energy tends to occur when more computer parts are used other than just computations done by the CPU. This cycle can be seen more expressively in Figure 7.6 because this modeling is much lengthier than the other experiments. So, it requires more computer resources that are not exclusively the CPU.

The experimental results indicated that different classification methods, including the Naïve Bayes, KNN, Hoeffding Tree, and Oza Bagging, produced different energy footprints, as indicated in Figures 7.3, 7.4, 7.5 and 7.6.

The total amount of energy consumed for each experiment can be calculated to indicate the efficiency of each method and can be seen in the Table 7.1. The graphs show that each classification method took a different time to complete, with distinct energy footprints. As expected, the results show that the type of classification method chosen impacts the energy consumption during the course of the experiment. Consequently, it is possible to determine which model will use more resources. However, because each model has many variables, the attempt to model the energy consumption of each particular model is a task not covered by this work.

This experiment was able to answer how energy consumption can be compared against different classifiers. Each classifier used consumed different amounts of energy,

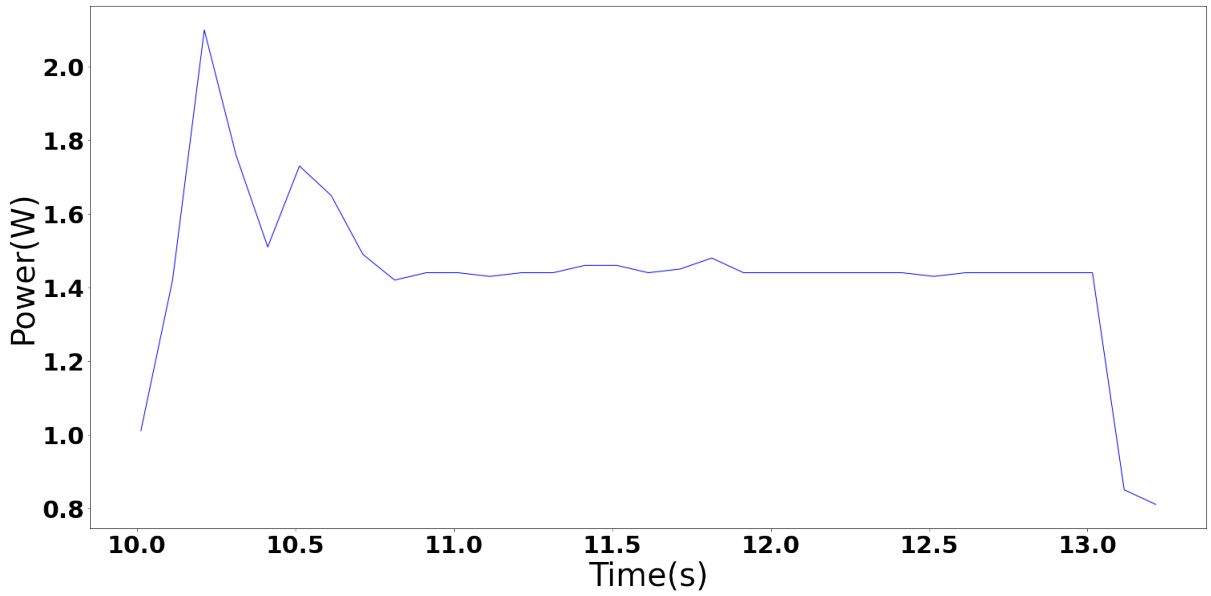

Figure 7.3: Experiment 2's results for Naïve Bayes. The graph shows the power consumption over time for the second experiment. This modeling is very short, so it only shows a peak of consumption during the initial stage of the experiment.

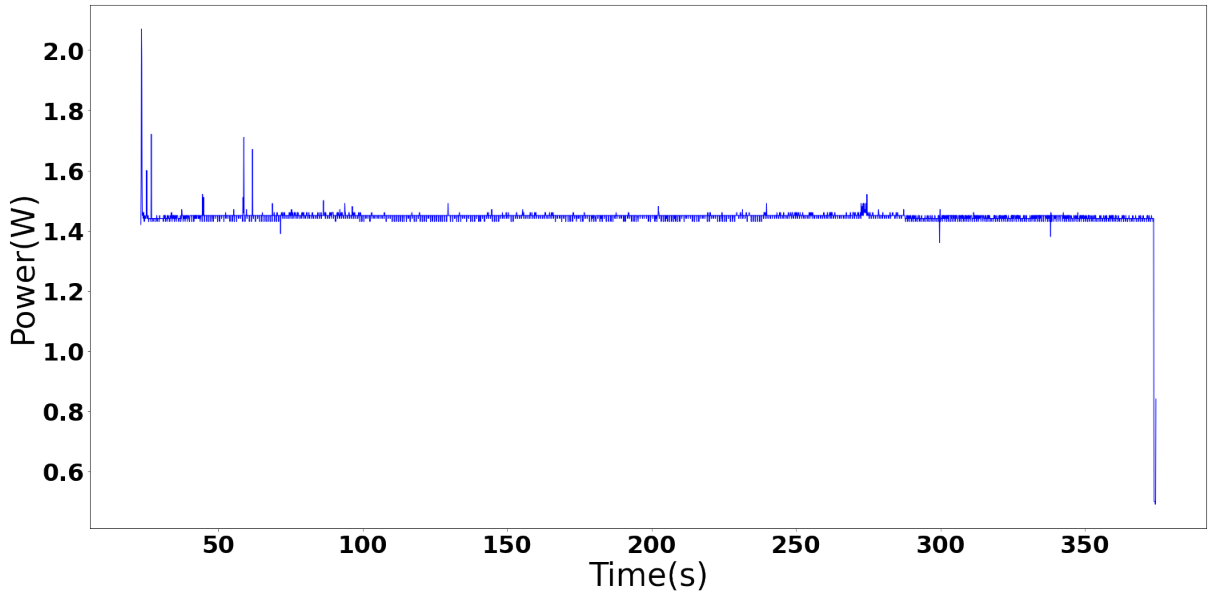

Figure 7.4: Experiment 2's results for kNN. The graph shows the power consumption over time for the second experiment. Aside from the very short initial peak consumption, this experiment shows some peaks of consumption, that are not very expressive. This type of small peaks can be considered noise. The initial peak consumption can be considered low due to the fact that kNN is a very simple model and requires almost no setup.

having different degrees of final accuracy.

Each classifier had different degrees of performance and energy consumption. Therefore, the power consumption over time can give the total consumption of each. For ex-

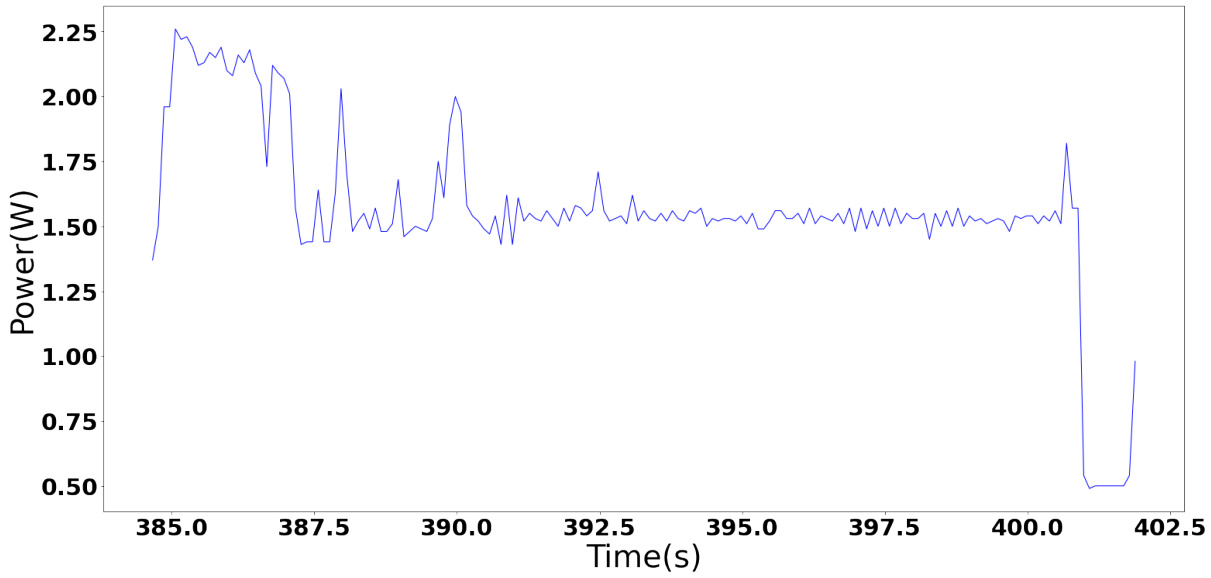

Figure 7.5: Experiment 2's results for Hoeffding Tree. The graph shows the power consumption over time for the second experiment. Here again we see the initial peak consumption, some more consumption at the initial stages, but does not show much difference as the experiment continues.

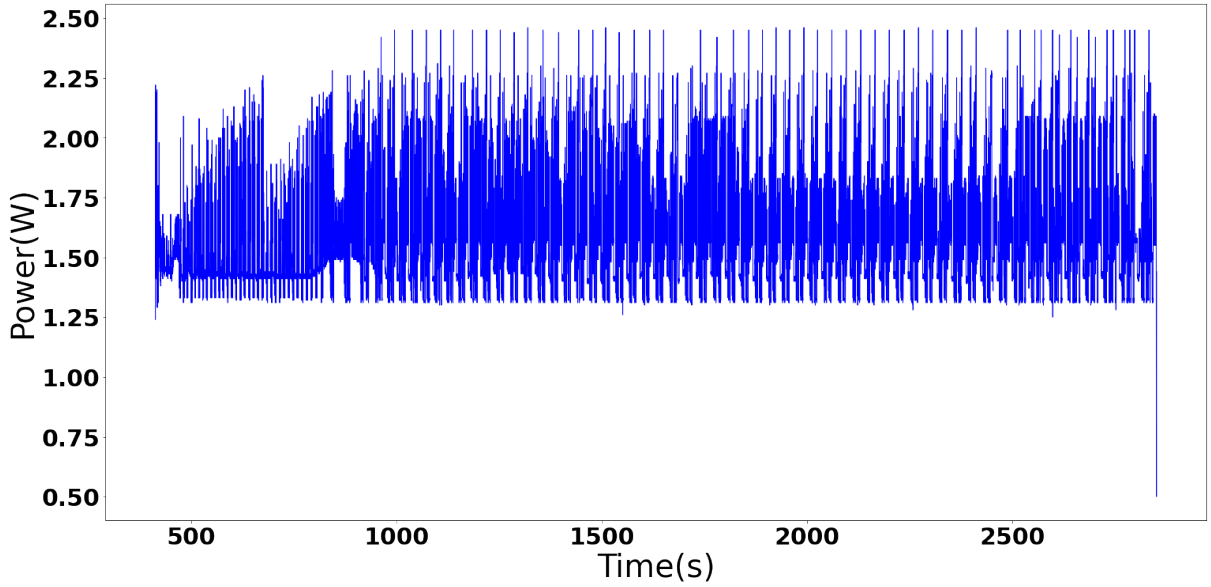

Figure 7.6: Experiment 2's results for Oza Bagging. The graph shows the power consumption over time for the second experiment. In this experiment, the cycles are clearly visible, with highs and lows in energy consumption, due to the large amount of resources required by this model. When there is a peak of energy, is when the computer is utilizing more resources that are not CPU, and the lows are when only a part of the resources are being utilized.

ample, the simplest one, Naïve Bayes, spent less than 3 seconds' worth of time to provide a model. In contrast, Oza Bagging, an ensemble, spent almost 2500 seconds. The time

used by each classifier is a large part of the energy, especially when one considers the static power consumed by the computer. Each classifier takes a different time, mainly depending on its internal workings, code, and the number of calculations done by each model. Also, each classifier has a different performance at the end of the calculations that may differ significantly.

7.3 Experiment 3 - Comparison of energy consumption in prequential and interleaved chunks validation schemes

The third experiment that was conducted to determine energy consumption in data stream mining involved using different evaluation methods (prequential, and interleaved chucks) and to compare if they have the same or different energy footprints. The plugin developed for this dissertation previously described in Section 6.7 was used to acquire the energy measurements for this experiment.

For all runs, all parameters were kept equal except the evaluation methods and the model. Utilizing the same setup for the computer as mentioned in Section 7, the main difference between this experiment and the experiment from Section 7.2 is the tool to measure energy consumption. An external software tool was used to measure consumption in the previous experiment. In contrast, the plugin developed for this dissertation was used in this experiment. A script was written to execute the experiments and output the results. The plugin works by measuring the consumption during the experiment and outputting the values and results. Each experiment ran five times, and the averages were taken. Between each run for the same parameters, there was little change in the results for the same model/evaluator pair, as seen in Table 7.2.

Since in this experiment the goal was to evaluate the energetic impact of the prequential compared to interleaved chunks, it was attempted to find a trend of either an increased or decreased energy consumption when comparing one method to the other.

The results were not clear whether the evaluation method increases or decreases the energy consumption. As can been seen, in some classification methods, the energy footprint increased, while for other methods it decreased. The results of the experiment are summarized in the Table 7.1.

When looking closely, the total energy consumed by the system directly relates to the total time spent. For example, ARF consumed 45,09kJ when using prequential with over a period of 979 seconds. At the same time, when using interleaved chunks,

Table 7.1: Experiment 3's results. This table represents the average results from Table 7.2.

Classifier	Evaluation	Energy(J)	Avg Acc(%)	Final Acc(%)	Time
ARF	Prequential	45096,38	12,15	14,50	979
	Interleaved	47505,18	12,30	14,38	1027
Hoeffding Tree	Prequential	398,58	31,72	39,10	26
	Interleaved	480,91	13,73	16,21	27
kNN	Prequential	4670,92	12,82	15,60	322
	Interleaved	4677,16	10,22	10,98	313
Naïve Bayes	Prequential	129,83	16,04	19,10	9
	Interleaved	124,92	15,97	16,11	8
Oza Bagging	Prequential	48166,66	32,88	41,00	3134
	Interleaved	50284,18	14,02	16,67	3312

it consumed 47,50KJ and was completed in a period of 1027 seconds. The average consumption of the experiment is 46W for both prequential and interleaved chunks. In all experiments, it seems that the potency consumed by the computer is roughly the same for both interleaved chunks and prequential for the same model.

Table 7.2: Experiment 3: The complete results for the runs. In this table, for each classifier/evaluator pair, the average results of total energy consumed, average accuracy, final accuracy and total CPU time for each run is presented.

Classifier	Evaluator	Run No.	Energy(J)	Avg Acc(%)	Final Acc(%)	Time(s)
ARF	Interleaved	1	47483,62	12,3	14,38	962,46
ARF	Interleaved	2	47984,17	12,3	14,38	1009,23
ARF	Interleaved	3	46632,25	12,3	14,38	950,75
ARF	Interleaved	4	47942,69	12,3	14,38	1010,11
ARF	Interleaved	5	47483,2	12,3	14,38	963,08
ARF	Prequential	1	45291,33	12,16	14,5	1035,59
ARF	Prequential	2	45968,76	12,16	14,5	1062,75
ARF	Prequential	3	45537,14	12,16	14,5	1035,65
ARF	Prequential	4	43342,31	12,16	14,5	956,34
ARF	Prequential	5	45342,36	12,16	14,5	1046,8
Hoeffding Tree	Interleaved	1	484,5	13,73	16,22	27,3
Hoeffding Tree	Interleaved	2	476,1	13,73	16,22	26,6
Hoeffding Tree	Interleaved	3	484,35	13,73	16,22	26,92
Hoeffding Tree	Interleaved	4	479,11	13,73	16,22	27,32
Hoeffding Tree	Interleaved	5	480,52	13,73	16,22	27,11
Hoeffding Tree	Prequential	1	393,06	31,73	39,1	26,41
Hoeffding Tree	Prequential	2	394,98	31,73	39,1	26,43

Table 7.2: Experiment 3: The complete results for the runs. In this table, for each classifier/evaluator pair, the average results of total energy consumed, average accuracy, final accuracy and total CPU time for each run is presented.

Classifier	Evaluator	Run No.	Energy(J)	Avg Acc(%)	Final Acc(%)	Time(s)
Hoeffding Tree	Prequential	3	400,48	31,73	39,1	26,7
Hoeffding Tree	Prequential	4	396,74	31,73	39,1	26,44
Hoeffding Tree	Prequential	5	407,67	31,73	39,1	27,38
kNN	Interleaved	1	4699,34	10,22	10,98	314,88
kNN	Interleaved	2	4668,05	10,22	10,98	312,63
kNN	Interleaved	3	4666,87	10,22	10,98	312,51
kNN	Interleaved	4	4672,89	10,22	10,98	312,97
kNN	Interleaved	5	4678,66	10,22	10,98	313,19
kNN	Prequential	1	4640,13	12,83	15,6	320,8
kNN	Prequential	2	4737,05	12,83	15,6	326,91
kNN	Prequential	3	4645,92	12,83	15,6	320,7
kNN	Prequential	4	4582,77	12,83	15,6	315,96
kNN	Prequential	5	4748,75	12,83	15,6	327,14
Naïve Bayes	Interleaved	1	125,1	15,98	16,12	8,81
Naïve Bayes	Interleaved	2	124,8	15,98	16,12	8,79
Naïve Bayes	Interleaved	3	124,83	15,98	16,12	8,82
Naïve Bayes	Interleaved	4	124,98	15,98	16,12	8,81
Naïve Bayes	Interleaved	5	124,89	15,98	16,12	8,79
Naïve Bayes	Prequential	1	129,53	16,04	19,1	9,14
Naïve Bayes	Prequential	2	130,39	16,04	19,1	9,17
Naïve Bayes	Prequential	3	129,47	16,04	19,1	9,16
Naïve Bayes	Prequential	4	129,73	16,04	19,1	9,15
Naïve Bayes	Prequential	5	130,04	16,04	19,1	9,16
Oza Bagging	Interleaved	1	49638,17	14,03	16,68	3265,18
Oza Bagging	Interleaved	2	50876,19	14,03	16,68	3358,21
Oza Bagging	Interleaved	3	49392,51	14,03	16,68	3240,17
Oza Bagging	Interleaved	4	50902,85	14,03	16,68	3354,89
Oza Bagging	Interleaved	5	50611,19	14,03	16,68	3342,44
Oza Bagging	Prequential	1	46887,27	32,89	41,0	3055,98
Oza Bagging	Prequential	2	49362,86	32,89	41,0	3203,99
Oza Bagging	Prequential	3	48931,54	32,89	41,0	3185,79
Oza Bagging	Prequential	4	47297,81	32,89	41,0	3074,57
Oza Bagging	Prequential	5	48353,86	32,89	41,0	3152,25

Table 7.3: Experiment 3: Standard Deviation (Std Dev) and Coefficient of Variation (CV) between the runs. As can be seen, the CV is fairly small for all cases, lower than 3% for all cases.

Classifier	Evaluator	Run No.	Energy(J)	Std Dev	Coeff Var(%)
ARF	Interleaved	1	47483,62	544,01	1,14516578
ARF	Interleaved	2	47984,17		
ARF	Interleaved	3	46632,25		
ARF	Interleaved	4	47942,69		
ARF	Interleaved	5	47483,2		
ARF	Prequential	1	45291,33	1016,19	2,253379504
ARF	Prequential	2	45968,76		
ARF	Prequential	3	45537,14		
ARF	Prequential	4	43342,31		
ARF	Prequential	5	45342,36		
Hoeffding Tree	Interleaved	1	484,5	3,579	0,744297442
Hoeffding Tree	Interleaved	2	476,1		
Hoeffding Tree	Interleaved	3	484,35		
Hoeffding Tree	Interleaved	4	479,11		
Hoeffding Tree	Interleaved	5	480,52		
Hoeffding Tree	Prequential	1	393,06	5,76	1,446977577
Hoeffding Tree	Prequential	2	394,98		
Hoeffding Tree	Prequential	3	400,48		
Hoeffding Tree	Prequential	4	396,74		
Hoeffding Tree	Prequential	5	407,67		
kNN	Interleaved	1	4699,34	13,24	0,283102084
kNN	Interleaved	2	4668,05		
kNN	Interleaved	3	4666,87		
kNN	Interleaved	4	4672,89		
kNN	Interleaved	5	4678,66		
kNN	Prequential	1	4640,13	70,31	1,505275251
kNN	Prequential	2	4737,05		
kNN	Prequential	3	4645,92		
kNN	Prequential	4	4582,77		
kNN	Prequential	5	4748,75		

Naïve Bayes	Interleaved	1	125,1	0,12	0,097550893
Naïve Bayes	Interleaved	2	124,8		
Naïve Bayes	Interleaved	3	124,83		
Naïve Bayes	Interleaved	4	124,98		
Naïve Bayes	Interleaved	5	124,89		
Naïve Bayes	Prequential	1	129,53	0,38	0,295128304
Naïve Bayes	Prequential	2	130,39		
Naïve Bayes	Prequential	3	129,47		
Naïve Bayes	Prequential	4	129,73		
Naïve Bayes	Prequential	5	130,04		
Oza Bagging	Interleaved	1	49638,17	716,33	1,42458189
Oza Bagging	Interleaved	2	50876,19		
Oza Bagging	Interleaved	3	49392,51		
Oza Bagging	Interleaved	4	50902,85		
Oza Bagging	Interleaved	5	50611,19		
Oza Bagging	Prequential	1	46887,27	1053,88	2,188002492
Oza Bagging	Prequential	2	49362,86		
Oza Bagging	Prequential	3	48931,54		
Oza Bagging	Prequential	4	47297,81		
Oza Bagging	Prequential	5	48353,86		

The coefficient of variation is the relative dispersion that a value has compared to the mean value of a particular population. For instance, suppose the standard deviation is the absolute dispersion from the mean. Then, the Coefficient of Variation is the relative dispersion from the mean. It is calculated by dividing the standard deviation σ by the mean in percentage, as described in Equation 7.1.

$$CV\% = \frac{100\sigma}{mean} \tag{7.1}$$

Table 7.1 indicates that the energy footprint increases in some cases. In contrast, it decreases in others when the evaluation method is changed. As seen on Table 7.3, the values of the standard deviation are fairly small when compared to the mean, and the Coefficient of Variation (BROWN, 1998) is lower than 3% for all cases. This result implies that it is possible to know why there is an increase or decrease in each situation (meaning it is not random). However, it is impossible to generalize a rule to whether

prequential consume more or less energy than interleaved chunks without a more robust mathematical equation for each model.

Consequently, this experimentation depicts different classifiers exhibiting different energy consumption footprints under different validation schemes. Also, it verifies that there is no trend in the increase or decrease of energy consumption, meaning there is no direct relationship between the change of evaluation method and the energy consumption. Thus, this is a relatively strong indication that an energy meter would be helpful during the model's development phase.

When looking closely at each run depicted in Table 7.2, one can notice that the average accuracy and final accuracy are the same for the same experiment. This happens because the seed used for the experiment is the same for all runs. Thus, it is expected that the results will yield the same values. The seed values must be the same, as we want to measure the difference in consumption with the same values. Therefore, having different seeds could potentially net different results in the final model, consequently in the energy measurements.

One interesting result taken from this experiment is that in all cases, prequential generated a more accurate final result than interleaved chunks. Even if small, the time and energy difference indicates that interleaved chunks produces results more quickly. In some cases, the difference is accuracy is so small that is makes sense to choose interleaved chunks evaluation, while on others, namely Oza Bagging and Hoeffding Tree, the final accuracy of the model is wildly different (41% against 16,67% for Oza Bagging and 39,10% against 16,21% for Hoeffding Tree).

In contrast, while it has been shown that prequential does yield better final accuracy for all cases, interleaved chunks is a good contender in some cases, while maintaining a marginally better energy footprint.

7.4 Experiment 4 - Comparison of energy consumption with and without concept drift

The last experiment conducted to determine energy consumption in data stream mining involved determining if the concept drift to an easier to classify concept would yield lower or higher energy consumption. Concept drift is the shift in the underlying distribution of examples arriving from a data stream. In fact, in data stream mining, the capacity to detect and adapt to change in the distribution of examples is essential for learning algorithms. The concept drift occurs over time, and the rate at which it occurs

may vary. The results from the experiment were summarized and are given in Table 7.4.

Again, in this final experiment, all parameters for all runs were the same, changing evaluation methods, model, and lastly, the input containing a concept drift and not containing a drift. The concept drift utilized is much easier to learn than the initial concept, having less classes and easier parameters. Utilizing the same setup mentioned in Section 7 and again using the plugin developed for this dissertation. A script was used to automate the runs and save the results. Similarly to the previous section results, little change could be seen between the experiments, as seen in Table 7.5. For example, between runs 1 through 5 of Naïve Bayes, using interleaved chunks as a validator and no concept drift, the standard deviation is 0.121861 and the coefficient of variation is 0.097551%. The experiment that generated the higher coefficient of variation is Naïve Bayes, with interleaved chunks and having concept drift, that generated a standard deviation of 11,94976 and a coefficient of variation of 6,880333%. All these results can be seen on Table 7.6.

Table 7.4: Experiment 4's average results. While some results indicate that an easier concept drift increase the energy consumption on some classifiers, while on others it decreases the consumption.

Classifier	Evaluation	Drift	Avg Acc(%)	Final Acc(%)	Energy(J)	Time(s)
Naive Bayes	Prequential	No	16,04	15,97	129,83	9
		Yes	45,82	21,21	164,48	10
	Interleaved Chunks	No	15,97	16,11	124,92	9
		Yes	21,21	45,72	173,68	10
kNN	Prequential	No	12,82	15,60	4670,92	322
		Yes	46,57	85,60	4839,85	329
	Interleaved Chunks	No	10,22	10,98	4677,16	313
		Yes	21,44	46,49	4569,18	306
Hoeffding Tree	Prequential	No	31,72	39,10	398,58	26
		Yes	56,16	99,90	382,80	20
	Interleaved Chunks	No	13,73	16,21	480,91	27
		Yes	25,88	55,84	387,01	20
ARF	Prequential	No	12,15	14,50	45096,38	1027
		Yes	55,02	100	61251,09	3895
	Interleaved Chunks	No	12,30	14,38	47505,18	979
		Yes	23,94	54,84	60362,56	3843
Oza Bagging	Prequential	No	32,88	41,00	48166,66	3134
		Yes	56,32	100	36490,10	2395
	Interleaved Chunks	No	14,02	16,67	50284,18	3312
		Yes	26,07	56,02	37088,83	2443

Table 7.5: Experiment 4's complete results. Some names were shortened because the table was too large. All accuracies, both average and final were exactly the same throughout all runs for the same parameters. The only values that have a slight change are the energy consumption and the time, though as seen in Table 5.6.

Classifier	Eval	Drift	Run	Energy	Avg.Acc.	Final.Acc.	Time
ARF	Int.Ch.	No	1	47483.62	12.3	14.38	962.46
ARF	Int.Ch.	No	2	47984.17	12.3	14.38	1009.23
ARF	Int.Ch.	No	3	46632.25	12.3	14.38	950.75
ARF	Int.Ch.	No	4	47942.69	12.3	14.38	1010.11
ARF	Int.Ch.	No	5	47483.2	12.3	14.38	963.08
ARF	Int.Ch.	Yes	1	59911.95	23.94	54.84	3814.9
ARF	Int.Ch.	Yes	2	60203.79	23.94	54.84	3835.04
ARF	Int.Ch.	Yes	3	61402.9	23.94	54.84	3910.79
ARF	Int.Ch.	Yes	4	60110.36	23.94	54.84	3825.74
ARF	Int.Ch.	Yes	5	60183.82	23.94	54.84	3832.85
ARF	Preq	No	1	45291.33	12.16	14.5	1035.59
ARF	Preq	No	2	45968.76	12.16	14.5	1062.75
ARF	Preq	No	3	45537.14	12.16	14.5	1035.65
ARF	Preq	No	4	43342.31	12.16	14.5	956.34
ARF	Preq	No	5	45342.36	12.16	14.5	1046.8
ARF	Preq	Yes	1	62254.79	55.02	100.0	3963.9
ARF	Preq	Yes	2	60838.82	55.02	100.0	3875.08
ARF	Preq	Yes	3	60771.51	55.02	100.0	3859.78
ARF	Preq	Yes	4	61888.61	55.02	100.0	3943.04
ARF	Preq	Yes	5	60501.75	55.02	100.0	3835.06
Hoeffding Tree	Int.Ch.	No	1	484.5	13.73	16.22	27.3
Hoeffding Tree	Int.Ch.	No	2	476.1	13.73	16.22	26.6
Hoeffding Tree	Int.Ch.	No	3	484.35	13.73	16.22	26.92
Hoeffding Tree	Int.Ch.	No	4	479.11	13.73	16.22	27.32
Hoeffding Tree	Int.Ch.	No	5	480.52	13.73	16.22	27.11
Hoeffding Tree	Int.Ch.	Yes	1	384.0	25.88	55.85	19.82
Hoeffding Tree	Int.Ch.	Yes	2	383.12	25.88	55.85	19.99
Hoeffding Tree	Int.Ch.	Yes	3	394.43	25.88	55.85	20.48
Hoeffding Tree	Int.Ch.	Yes	4	385.46	25.88	55.85	19.96
Hoeffding Tree	Int.Ch.	Yes	5	388.07	25.88	55.85	19.82

Table 7.5: Experiment 4's complete results. Some names were shortened because the table was too large. All accuracies, both average and final were exactly the same throughout all runs for the same parameters. The only values that have a slight change are the energy consumption and the time, though as seen in Table 5.6.

Classifier	Eval	Drift	Run	Energy	Avg.Acc.	Final.Acc.	Time
Hoeffding Tree	Preq	No	1	393.06	31.73	39.1	26.41
Hoeffding Tree	Preq	No	2	394.98	31.73	39.1	26.43
Hoeffding Tree	Preq	No	3	400.48	31.73	39.1	26.7
Hoeffding Tree	Preq	No	4	396.74	31.73	39.1	26.44
Hoeffding Tree	Preq	No	5	407.67	31.73	39.1	27.38
Hoeffding Tree	Preq	Yes	1	379.12	56.17	99.9	20.13
Hoeffding Tree	Preq	Yes	2	381.41	56.17	99.9	19.95
Hoeffding Tree	Preq	Yes	3	388.39	56.17	99.9	20.04
Hoeffding Tree	Preq	Yes	4	384.24	56.17	99.9	20.06
Hoeffding Tree	Preq	Yes	5	380.87	56.17	99.9	20.07
kNN	Int.Ch.	No	1	4699.34	10.22	10.98	314.88
kNN	Int.Ch.	No	2	4668.05	10.22	10.98	312.63
kNN	Int.Ch.	No	3	4666.87	10.22	10.98	312.51
kNN	Int.Ch.	No	4	4672.89	10.22	10.98	312.97
kNN	Int.Ch.	No	5	4678.66	10.22	10.98	313.19
kNN	Int.Ch.	Yes	1	4573.87	21.44	46.49	306.64
kNN	Int.Ch.	Yes	2	4545.19	21.44	46.49	305.99
kNN	Int.Ch.	Yes	3	4565.17	21.44	46.49	306.65
kNN	Int.Ch.	Yes	4	4595.76	21.44	46.49	308.91
kNN	Int.Ch.	Yes	5	4565.94	21.44	46.49	306.37
kNN	Preq	No	1	4640.13	12.83	15.6	320.8
kNN	Preq	No	2	4737.05	12.83	15.6	326.91
kNN	Preq	No	3	4645.92	12.83	15.6	320.7
kNN	Preq	No	4	4582.77	12.83	15.6	315.96
kNN	Preq	No	5	4748.75	12.83	15.6	327.14
kNN	Preq	Yes	1	4847.95	46.58	85.6	330.85
kNN	Preq	Yes	2	4852.7	46.58	85.6	331.09
kNN	Preq	Yes	3	4863.45	46.58	85.6	331.18
kNN	Preq	Yes	4	4881.05	46.58	85.6	332.04
kNN	Preq	Yes	5	4754.12	46.58	85.6	324.03

Table 7.5: Experiment 4's complete results. Some names were shortened because the table was too large. All accuracies, both average and final were exactly the same throughout all runs for the same parameters. The only values that have a slight change are the energy consumption and the time, though as seen in Table 5.6.

Classifier	Eval	Drift	Run	Energy	Avg.Acc.	Final.Acc.	Time
Naïve Bayes	Int.Ch.	No	1	125.1	15.98	16.12	8.81
Naïve Bayes	Int.Ch.	No	2	124.8	15.98	16.12	8.79
Naïve Bayes	Int.Ch.	No	3	124.83	15.98	16.12	8.82
Naïve Bayes	Int.Ch.	No	4	124.98	15.98	16.12	8.81
Naïve Bayes	Int.Ch.	No	5	124.89	15.98	16.12	8.79
Naïve Bayes	Int.Ch.	Yes	1	170.18	21.22	45.72	9.93
Naïve Bayes	Int.Ch.	Yes	2	163.47	21.22	45.72	9.95
Naïve Bayes	Int.Ch.	Yes	3	170.57	21.22	45.72	9.95
Naïve Bayes	Int.Ch.	Yes	4	194.41	21.22	45.72	9.94
Naïve Bayes	Int.Ch.	Yes	5	169.77	21.22	45.72	9.94
Naïve Bayes	Preq	No	1	129.53	16.04	19.1	9.14
Naïve Bayes	Preq	No	2	130.39	16.04	19.1	9.17
Naïve Bayes	Preq	No	3	129.47	16.04	19.1	9.16
Naïve Bayes	Preq	No	4	129.73	16.04	19.1	9.15
Naïve Bayes	Preq	No	5	130.04	16.04	19.1	9.16
Naïve Bayes	Preq	Yes	1	165.01	45.83	88.1	10.31
Naïve Bayes	Preq	Yes	2	164.6	45.83	88.1	10.28
Naïve Bayes	Preq	Yes	3	162.74	45.83	88.1	10.31
Naïve Bayes	Preq	Yes	4	167.05	45.83	88.1	10.32
Naïve Bayes	Preq	Yes	5	163.02	45.83	88.1	10.27
Oza Bagging	Int.Ch.	No	1	49638.17	14.03	16.68	3265.18
Oza Bagging	Int.Ch.	No	2	50876.19	14.03	16.68	3358.21
Oza Bagging	Int.Ch.	No	3	49392.51	14.03	16.68	3240.17
Oza Bagging	Int.Ch.	No	4	50902.85	14.03	16.68	3354.89
Oza Bagging	Int.Ch.	No	5	50611.19	14.03	16.68	3342.44
Oza Bagging	Int.Ch.	Yes	1	36705.2	26.08	56.03	2411.22
Oza Bagging	Int.Ch.	Yes	2	37744.87	26.08	56.03	2490.1
Oza Bagging	Int.Ch.	Yes	3	36345.82	26.08	56.03	2401.82
Oza Bagging	Int.Ch.	Yes	4	36876.62	26.08	56.03	2434.47
Oza Bagging	Int.Ch.	Yes	5	37771.69	26.08	56.03	2477.83

Table 7.5: Experiment 4's complete results. Some names were shortened because the table was too large. All accuracies, both average and final were exactly the same throughout all runs for the same parameters. The only values that have a slight change are the energy consumption and the time, though as seen in Table 5.6.

Classifier	Eval	Drift	Run	Energy	Avg.Acc.	Final.Acc.	Time
Oza Bagging	Preq	No	1	46887.27	32.89	41.0	3055.98
Oza Bagging	Preq	No	2	49362.86	32.89	41.0	3203.99
Oza Bagging	Preq	No	3	48931.54	32.89	41.0	3185.79
Oza Bagging	Preq	No	4	47297.81	32.89	41.0	3074.57
Oza Bagging	Preq	No	5	48353.86	32.89	41.0	3152.25
Oza Bagging	Preq	Yes	1	36161.12	56.32	100.0	2375.08
Oza Bagging	Preq	Yes	2	36809.74	56.32	100.0	2412.67
Oza Bagging	Preq	Yes	3	36819.92	56.32	100.0	2423.36
Oza Bagging	Preq	Yes	4	36310.88	56.32	100.0	2386.77
Oza Bagging	Preq	Yes	5	36348.88	56.32	100.0	2381.99

Table 7.6: Experiment 4: Standard Deviation (Std Dev) and Coefficient of Variation (CV) between the runs.

Classifier	Evaluator	Drift	Run No.	Energy(J)	Std.Dev.	CV(%)
ARF	Int.Ch.	No	1	47483,62	544,0131	1,145166
			2	47984,17		
			3	46632,25		
			4	47942,69		
			5	47483,2		
ARF	Int.Ch.	Yes	1	59911,95	592,8991	0,98223
			2	60203,79		
			3	61402,9		
			4	60110,36		
			5	60183,82		
ARF	Preq	No	1	45291,33	1016,193	2,25338
			2	45968,76		
			3	45537,14		
			4	43342,31		
			5	45342,36		
ARF	Preq	Yes	1	62254,79	770,6036	1,258106

Table 7.6: Experiment 4: Standard Deviation (Std Dev) and Coefficient of Variation (CV) between the runs.

Classifier	Evaluator	Drift	Run No.	Energy(J)	Std.Dev.	CV(%)
			2	60838,82		
			3	60771,51		
			4	61888,61		
			5	60501,75		
			1	484,5		
			2	476,1		
Hoeffding Tree	Int.Ch.	No	3	484,35	3,579445	0,744297
			4	479,11		
			5	480,52		
			1	384		
			2	383,12		
Hoeffding Tree	Int.Ch.	Yes	3	394,43	4,549025	1,17541
			4	385,46		
			5	388,07		
			1	393,06		
			2	394,98		
Hoeffding Tree	Preq	No	3	400,48	5,76745	1,446978
			4	396,74		
			5	407,67		
			1	379,12		
			2	381,41		
Hoeffding Tree	Preq	Yes	3	388,39	3,623566	0,94658
			4	384,24		
			5	380,87		
			1	4699,34		
			2	4668,05		
kNN	Int.Ch.	No	3	4666,87	13,24114	0,283102
			4	4672,89		
			5	4678,66		
			1	4573,87		
			2	4545,19		
kNN	Int.Ch.	Yes	3	4565,17	18,23863	0,399166

Table 7.6: Experiment 4: Standard Deviation (Std Dev) and Coefficient of Variation (CV) between the runs.

Classifier	Evaluator	Drift	Run No.	Energy(J)	Std.Dev.	CV(%)
			4	4595,76		
			5	4565,94		
kNN	Preq	No	1	4640,13	70,31026	1,505275
			2	4737,05		
			3	4645,92		
			4	4582,77		
			5	4748,75		
kNN	Preq	Yes	1	4847,95	49,58519	1,024518
			2	4852,7		
			3	4863,45		
			4	4881,05		
			5	4754,12		
Naïve Bayes	Int.Ch.	No	1	125,1	0,121861	0,097551
			2	124,8		
			3	124,83		
			4	124,98		
			5	124,89		
Naïve Bayes	Int.Ch.	Yes	1	170,18	11,94976	6,880333
			2	163,47		
			3	170,57		
			4	194,41		
			5	169,77		
Naïve Bayes	Preq	No	1	129,53	0,383171	0,295128
			2	130,39		
			3	129,47		
			4	129,73		
			5	130,04		
Naïve Bayes	Preq	Yes	1	165,01	1,736327	1,05562
			2	164,6		
			3	162,74		
			4	167,05		

Table 7.6: Experiment 4: Standard Deviation (Std Dev) and Coefficient of Variation (CV) between the runs.

Classifier	Evaluator	Drift	Run No.	Energy(J)	Std.Dev.	CV(%)
			5	163,02		
Oza Bagging	Int.Ch.	No	1	49638,17	716,3394	1,424582
			2	50876,19		
			3	49392,51		
			4	50902,85		
			5	50611,19		
Oza Bagging	Int.Ch.	Yes	1	36705,2	640,4989	1,726932
			2	37744,87		
			3	36345,82		
			4	36876,62		
			5	37771,69		
Oza Bagging	Preq	No	1	46887,27	1053,888	2,188002
			2	49362,86		
			3	48931,54		
			4	47297,81		
			5	48353,86		
Oza Bagging	Preq	Yes	1	36161,12	304,648	0,834878
			2	36809,74		
			3	36819,92		
			4	36310,88		
			5	36348,88		

As previously stated, concept drift is an essential aspect in data stream mining. During the experiment, it was evident that some of the classifiers increased energy consumption while others decreased energy consumption. For example, while Naïve Bayes increased the energy consumption by a tiny amount with the easier concept, Oza Bagging greatly decreased the energy consumption with the drift (see Table 7.4). Even though it is possible to establish a reason as to why specific classifiers increased their energy consumption, it seems not feasible to establish a relationship between energy consumption solely based on the existence of a concept drift or not.

As shown in Table 7.4, while a drift to a easier concept produces almost no change for some models on energy consumption, it does induce some changes in others. For exam-

ple, while ARF has increased energy consumption, Oza Bagging significantly decreased.

This behavior is perfectly explainable when knowing the inner workings of each classifier. For example, ARF creates a new model in parallel when it detects a concept drift, so it is understandable why it consumes more energy with an easier drift. At the same time, Oza Bagging will discard older (and potentially more complex) models, thus decreasing its consumption.

One crucial fact present in this experiment is the random seed that is shared across all runs. This produces the same model for each run, allowing for a more accurate comparison for the factor under study, drift, in this case.

Thus, even although for each specific classifier, it might be possible to determine if each variable will increase or decrease its energy consumption when isolated, when in a global state, when one wants to vary many variables at once, including the classifier, this study relates that there is enough empirical evidence to confirm that it is not possible to determine what all changes will impact in the overall energy consumption.

7.5 Discussion

The study's main objective is to propose a plugin to measure energy consumption in data stream mining. The study was motivated by the fact that it is hard to measure energy consumption during data stream mining because of a limited number of tools to measure energy consumption easily available to a data scientist. Thus, an accompanying study was performed to verify if a tool to measure energy consumption is relevant in data stream mining. Suppose the experiments' results showed that all variables could accurately predict whether the energy consumption would increase or decrease, the necessity of a tool to measure would drastically decrease. Furthermore, suppose it is impossible to determine most of the energy consumption aspects of data stream mining. In that case, there is a gap and a necessity to develop a plugin to measure this energy consumption.

Essentially, the experiment results wanted to investigate the energy consumption using various parameters in a data stream mining model's creation phase. The three main counterclaims were addressed which included, not necessarily a variable that reduces the consumption in one classifier will also reduce in another; the time and energy in a machine learning algorithm are highly correlated and thus more efficient algorithms in a time manner probably will consume less energy; it is complex to predict energy consumption and thus making impractical. In addition, although energy and time are highly correlated, measuring energy consumption can provide a unique summary on top of measuring time because in a production environment, most of the time, the model will be idle and consume

very little energy.

7.5.1 Hypotheses

Each experiment had a set of hypotheses that could be answered with their results.

7.5.1.1 Experiment One

- When compared to the software solution, the energy consumption result given by the hardware solution will be equal or highly similar.
  - A.: The results have not confirmed this hypothesis. The values were very different.
- If different, the results between the hardware and software will be correlated.
  - A.: The results have confirmed this hypothesis. The correlation between the results was very close to 1, indicating a very strong correlation.

7.5.1.2 Experiment Two

- The energy consumption of different data stream models is different, even when using the same inputs.
  - A.: The results have confirmed this hypothesis, as they were not even comparable in size and energy consumption.

7.5.1.3 Experiment Three

- The energy consumption of Interleaved Chunks will be lower than the consumption of Prequential because it processes more instances at a time, thus being more optimized.
  - A.: The results have not confirmed this hypothesis. For some configurations there was a consumption reduction, but not for all cases, thus falsifying this hypothesis.

7.5.1.4 Experiment Four

- When there is a concept drift to an easier concept, there will be a reduction in consumption because the model will require fewer resources to learn.
  - A.: The results have not confirmed this hypothesis. For some configurations there was a consumption reduction, but not for all cases, thus falsifying this hypothesis.

7.5.2 Concluding Remarks

There is a glaring question that might strike the reader that is: "does the act of measuring the energy using the same hardware that is performing the test will it impact the energy consumption?" The answer to that question is a simple yes. Measuring will increase the energy consumption and affect the results, but it was disregarded in this study for a few reasons.

The main reason is that the results given by a software solution do not consider the hardware's static power as it only considers the other processes' consumption. In light of that, the results can only be compared against experiments performed on the same hardware under the same conditions.

In addition to the above statement, the results can not be compared when performed in different conditions is the actual performance of each hardware. Hardware that is technologically more advanced tends to spend fewer resources to perform the same operations. Furthermore, even among the hardware carrying the exact specifications, each piece of equipment connected to that hardware will spend more energy. The differences in hardware, the operating system, installed software, kernel version, and others will impact the results.

Because of the reasons above, all measurements are performed using the same hardware and software, so the consumption impact will occur in the same proportion to all the tests, effectively nullifying their effects for comparison.

From the four experiments, it is evident that there are many possibilities of measuring data stream mining energy consumption. However, not all the possibilities are practical. For example, a hardware solution will require specific equipment. RAPL and PowerTOP require a complex setup that not every researcher is willing to acquire. Consequently, providing a plugin for facilitating researchers' energy consumption footprint experimenting helps in the data stream mining classifier selection process.

Chapter 8

Conclusions

This work drove the reader through a brief introduction to data stream mining and Green ICT, touching the topics of why data mining had a niche when it comes to data streams. It was also explained why it is essential to study this area in the ever-growing mountain of information generated in the world. To cope with a near-infinite and time-sensitive type of data, we explore different methods to extract information from this data, namely, data stream mining. As the name suggests, data stream mining focuses on extracting information from data streams, performing as well as their batch counterpart. However, some studies point out that one of the most common data stream mining methods, namely VFDT, tend to increase in size as more data arrives, posing a considerable problem in a data stream scenario.

Looking from the Green ICT prism, the constant growth of the data stream mining model, tending to infinity, is not desirable as it will spend more resources as time goes on. We briefly introduce the reader to some concepts concerning energy consumption, giving some thoughts on idle consumption and static power. We also touch on some of the more critical energy consumption topics, such as efficiency and CPU utilization.

As we have delved into the vast area of knowledge about Green ICT and energy consumption, the reader is invited to an energy-aware data stream mining assessment. In each of the four experiments, a new variable is added to experimentation. Initially, the first experiment compares software and hardware solutions to measure energy consumption. The results of this experiment allow the author to use a software solution, validating the possibility to gather energy consumption information directly from the processor without the required external hardware.

The second experiment was necessary to create the proposed framework and plugin for energy measurement. More specifically, this experiment measured the energy consumed by various classifiers under the same conditions. The measurement in this

experiment, done via an external software tool, while rather difficult, proved to be an invaluable asset while developing the plugin. Furthermore, it also showed that each classifier has a different energy footprint. While this allowed the author to determine a rank of which classifier was more energy-efficient, the approximate energy cost of each model can be compared roughly to its speed and execution time.

In the third and fourth experiments, it was compared the energy consumption in evaluation methods with and without concept drifts. In the third experiment, while it did tend to favor interleaved chunks as a more energy-efficient energy consumption method, it was unclear whether the drop in accuracy was worth the saving in some cases. Consequently, this analysis is required on each scenario in which a data scientist works on. On the other hand, the fourth experiment brought different results. In some cases, almost no change was detected. In others, it did increase or decrease the energy consumption by a considerate amount. Though this can be explained by how each classifier works internally, this is by no means trivial to model, and many variables may still be unknown when concerning energy consumption.

With this study, it is possible to confirm that the input data will impact the energy consumption of the data stream models. Furthermore, having another specific set of classifiers and validation protocols that all follow the same energy consumption trend[1] when presented with concept drift, will not change the conclusion. This reasoning is because the proposal and objective of this work are to find a generalized means to measure energy consumption with any configuration.

The researcher reckons that the data stream input generated for both experiments emphasizes the obtained results. Had the researcher chosen a more challenging concept drift (as opposed to an easier concept drift), there is a possibility that there would be an energy consumption increase in all cases. However, this possibility does not change the conclusion that not all models will increase their energy consumption by having a concept drift. Having one case that breaks this hypothesis is enough to say that it is incorrect. One have proven that by having a concept drift, not every model will increase its energy consumption, and on the same token, one also proved that not every model will decrease its energy consumption.

While this work aims towards measuring the total energy consumption of a computer during a data stream modeling process, it does not measure the individual energy consumption of each process. Also, a concise list of classifiers, evaluators, and drift detectors was chosen to enable the experiment. However, the author reckons that using other classifiers, evaluation methods and types of drift would provide better evidence of

[1]Either by increasing or decreasing by a specific value, be it absolute or relative

the findings. This work also does not provide a definite answer as to which model, evaluator, or drift detector is better, as this is the sole responsibility of the developer in a case-by-case manner.

Also, for this work, the energy consumption of the measurement process was disregarded because it is applied to all studied models.

This work is not aimed towards providing a definite mathematical modeling of each tested feature, as the main focus is evaluating the energy consumption.

All the results reinforce the need for an energy consumption measurement plugin that is easily accessible, due to the difficulty of obtaining the energy consumption without measuring.

Therefore, as a by-product of this research, the tool developed for assessing energy consumption is available as a plugin for the Massive Online Analysis (MOA) plugin.

There are, nonetheless, threats to the validity of this work. One threat to the validity of this work was when choosing the software solution to measure the consumption. Even though it was more straightforward when measuring via hardware, and the researcher had a more reassuring validation that it was the actual consumption given by the hardware, it was opted by the software solution for a couple of reasons:

- It is easier to distribute and replicate.
- It is more time accurate.
- Maintains the objective of this research in being a means to facilitate the measurement of energy consumption and a decision-making tool.

Though the researcher reckons the shortcomings of this approach, as it can only measure what is within the domain of the CPU, and though it is not an exact measure, it is proportional to the exact measure.

In order to mitigate this bias, the researcher took the following steps:

1. It used the same hardware and software for the measurements.
2. Performed an experiment that demonstrated the correlation between the hardware and software.
3. The environment was not modified between each trial.

Many future developments can be pursued, for example, on the metrics used to compare the results. This work used the sum of the total energy consumed by the process, but some metrics would be relevant, for example, the consumption per instance of the

input given to the model. That would be a suggested metric. One could have different input sizes and still properly compare the models. However, it is advisable to verify if the input change does not affect the models in an unknown manner. Another proposed metric worth pursuing is the energy spent to reach a certain accuracy level. It is known, for example, that there are diminishing returns when the accuracy rises, but the metric of energy spent to reach a certain degree of accuracy could be interesting. Potentially, setting a target accuracy and the experiment stops once it reaches that value, providing the researcher with the results.

Another suggestion for future works is measuring the model in states different from what is done in this work. For example, in this work, the researcher focused on the measurement during the development phase of the model, which means it is the cycle of predict $\rightarrow$ learn $\rightarrow$ evaluate. It would be interesting to verify the model's energy consumption when it is idle or when there is an interval of time between each instance.

A third possibility for future developments is to verify the impact of delays of the labels on the data. There is no delay between the instances and the labels in this work, but there might be significant differences if those are present.

Bibliography

ACCARDI, K. C.; YATES, A. *PowerTOP User's Guide*. [S.l.], 2014.

AKPAN, U.; AKPAN, G. The contribution of energy consumption to climate change:a feasible policy direction. *International Journal of Energy Economics and Policy*, v. 2, p. 21–33, 01 2011.

ALTMAN, N. S. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, Taylor & Francis, v. 46, n. 3, p. 175–185, 1992. Disponível em: <https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879>.

ANDERSON, J. M.; BERC, L. M.; DEAN, J.; GHEMAWAT, S.; HENZINGER, M. R.; LEUNG, S. T. A.; SITES, R. L.; VANDEVOORDE, M. T.; WALDSPURGER, C. A.; WEIHL, W. E. Continuous Profiling: Where Have All the Cycles Gone? *ACM Transactions on Computer Systems*, Association for Computing Machinery (ACM), v. 15, n. 4, p. 357–390, nov 1997. ISSN 07342071. Disponível em: <https://dl.acm.org/doi/10.1145/265924.265925>.

BAENA-GARCÍA, M.; Del Campo-´ Avila, J.; FIDALGO, R.; BIFET, A.; GAVALDÀ, R.; MORALES-BUENO, R. *Early Drift Detection Method*. [S.l.], 2005.

BARDDAL, J. P. *Feature Analysis in Evolving Data Streams : Issues and Algorithms*. Tese (Doutorado) — PUC-PR, 2019.

BARROS, R. S. M.; SANTOS, S. G. T. A large-scale comparison of concept drift detectors. *Information Sciences*, Elsevier Inc., v. 451-452, p. 348–370, jul. 2018. ISSN 00200255.

BARROSO, L. A.; HÖLZLE, U. The case for energy-proportional computing. *Computer*, v. 40, n. 12, p. 33–37, 2007.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, ACM PUB27 New York, NY, USA, v. 18, n. 9, p. 509–517, sep 1975. ISSN 15577317. Disponível em: <https://dl.acm.org/doi/abs/10.1145/361002.361007>.

BIFET, A.; GAVALDÁ, R. Learning from time-changing data with adaptive windowing. In: . [S.l.: s.n.], 2007. v. 7.

BIFET, A.; GAVALDÀ, R.; PFAHRINGER, B.; HOLMES, G. *Machine Learning for Data Streams with Practical Examples in MOA*. [S.l.: s.n.], 2018. ISBN 9780262037792.

BIFET, A.; HOLMES, G.; KIRKBY, R.; PFAHRINGER, B. Moa: Massive online analysis. *Journal of Machine Learning Research*, v. 11, n. 52, p. 1601–1604, 2010. Disponível em: <http://jmlr.org/papers/v11/bifet10a.html>.

BIFET, A.; HOLMES, G.; PFAHRINGER, B. Leveraging bagging for evolving data streams. In: BALCÁZAR, J. L.; BONCHI, F.; GIONIS, A.; SEBAG, M. (Ed.). *Machine Learning and Knowledge Discovery in Databases*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 135–150. ISBN 978-3-642-15880-3.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; GAVALDÀ, R. Improving adaptive bagging methods for evolving data streams. In: ZHOU, Z.-H.; WASHIO, T. (Ed.). *Advances in Machine Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 23–37. ISBN 978-3-642-05224-8.

BIFET, A.; KIRKBY, R. Data stream mining a practical approach. Citeseer, 2009.

BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

BOURDON, A.; NOUREDDINE, A.; ROUVOY, R.; SEINTURIER, L. PowerAPI: A Software Library to Monitor the Energy Consumed at the Process-Level. *ERCIM News*, v. 92, p. 43–44, 2013. Disponível em: <https://ercim-news.ercim.eu/en92/special/powerapi-a-software-library-to-monitor-the-energy-consumed-at-the-process-levelhttps://hal.inria.fr/hal-00850370{\%}5Cnhttps://hal.inria.fr/hal-00772>.

BREIMAN, L. *Classification and regression trees*. New York: Chapman & Hall, 1993. ISBN 978-0-412-04841-8.

BREIMAN, L. Bagging predictors. *Machine Learning*, v. 24, n. 2, p. 123–140, ago. 1996. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/BF00058655>.

BREIMAN, L. Random forests. *Machine Learning*, v. 45, n. 1, p. 5–32, out. 2001. ISSN 1573-0565. Disponível em: <https://doi.org/10.1023/A:1010933404324>.

BROWN, C. E. Coefficient of variation. In: _____. *Applied Multivariate Statistics in Geohydrology and Related Sciences*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. p. 155–157. ISBN 978-3-642-80328-4. Disponível em: <https://doi.org/10.1007/978-3-642-80328-4_13>.

CLIFTON, C. *Data mining*. ENCYCLOPÆDIA BRITANNICA, 2020. Disponível em: <https://www.britannica.com/technology/data-mining>.

COLLIER, D. The comparative method. In: _____. [S.l.: s.n.], 1993. p. 105–119.

COLMANT, M.; FELBER, P.; ROUVOY, R.; SEINTURIER, L.; WATTSKIT, L. S. *Software-Defined Power Monitoring of Distributed Systems*. [S.l.], 2017. 10 p. Disponível em: <http://powerapi.org>.

COLMANT, M.; HUERTAS, L.; KURPICZ, M.; ROUVOY, R.; FELBER, P.; SOBE, A. Process-level power estimation in VM-based systems. In: *Proceedings of the 10th European Conference on Computer Systems, EuroSys 2015*. New York, New York, USA: Association for Computing Machinery, Inc, 2015. p. 1–14. ISBN 9781450332385. Disponível em: <http://dl.acm.org/citation.cfm?doid=2741948.2741971>.

COLMANT, M.; ROUVOY, R.; KURPICZ, M.; SOBE, A.; FELBER, P.; SEINTURIER, L. The next 700 CPU power models. *Journal of Systems and Software*, Elsevier Inc., v. 144, p. 382–396, oct 2018. ISSN 01641212. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0164121218301377>.

DAVID, H.; GORBATOV, E.; HANEBUTTE, U. R.; KHANNA, R.; LE, C. RAPL: Memory power estimation and capping. In: *Proceedings of the International Symposium on Low Power Electronics and Design*. [S.l.: s.n.], 2010. p. 189–194. ISBN 9781450301466. ISSN 15334678.

DAVIS, H. P. Metering of electrical energy. *Transactions of the American Institute of Electrical Engineers*, v. 18, p. 277–285, 1901. ISSN 00963860.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. 2004.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*. [S.l.: s.n.], 2009.

DESROCHERS, S.; PARADIS, C.; WEAVER, V. M. A validation of dram rapl power measurements. In: *Proceedings of the Second International Symposium on Memory Systems*. New York, NY, USA: Association for Computing Machinery, 2016. (MEMSYS '16),

p. 455–470. ISBN 9781450343053. Disponível em: <https://doi.org/10.1145/2989081.2989088>.

DESROCHERS, S.; PARADIS, C.; WEAVER, V. M. *Reading RAPL energy measurements from Linux*. 2020. Http://web.eece.maine.edu/ vweaver/projects/rapl/. Accessed: 2020-10-08.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2000. (KDD '00), p. 71–80. ISBN 1581132336. Disponível em: <https://doi.org/10.1145/347090.347107>.

DU, L.; SONG, Q.; ZHU, L.; ZHU, X. A selective detector ensemble for concept drift detection. *Computer Journal*, Oxford University Press, v. 58, n. 3, p. 457–471, 2015. ISSN 14602067.

ELMIRGHANI, J. M.; KLEIN, T.; HINTON, K.; NONDE, L.; LAWEY, A. Q.; EL-GORASHI, T. E.; MUSA, M. O.; DONG, X. GreenTouch GreenMeter core network energy-efficiency improvement measures and optimization. *Journal of Optical Communications and Networking*, Institute of Electrical and Electronics Engineers Inc., v. 10, n. 2, p. A250–A269, feb 2018. ISSN 19430620.

Energy Magazine. *The Importance of Metering and Monitoring Energy Consumption | EM Magazine*. 2018. Disponível em: <https://www.energymanagermagazine.co.uk/the-importance-of-metering-and-monitoring-energy-consumption/>.

ENOKIDO, T.; AIKEBAIER, A.; TAKIZAWA, M. A model for reducing power consumption in peer-to-peer systems. *IEEE Systems Journal*, v. 4, n. 2, p. 221–229, jun. 2010. ISSN 19328184.

FABIAN, P.; MICHEL, V.; OLIVIERGRISEL, O. G.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; VANDERPLAS, J.; COURNAPEAU, D.; PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; THIRION, B.; GRISEL, O.; DUBOURG, V.; PASSOS, A.; BRUCHER, M.; ANDÉDOUARDAND, M. P.; DUCHESNAY andÉdouard; EDOUARDDUCHESNAY, F. D. *Scikit-learn: Machine Learning in Python*. [S.l.], 2011. v. 12, 2825–2830 p. Disponível em: <http://scikit-learn.sourceforge.net.>

FLINN, J.; SATYANARAYANAN, M. PowerScope: A tool for profiling the energy usage of mobile applications. In: *Proceedings - WMCSA'99: 2nd IEEE Workshop on Mobile Computing Systems and Applications*. [S.l.: s.n.], 1999. p. 1–9. ISBN 0769500250.

FRANK, E.; HALL, M. A.; WITTEN, I. H. *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", fourth edition ed. Morgan Kaufmann, 2016.* 2016.

GAMA, J. *Knowledge Discovery from Data Streams*. 1st. ed. [S.l.]: Chapman & Hall/CRC, 2010. ISBN 1439826110.

GAMA, J.; MEDAS, P.; CASTILLO, G.; RODRIGUES, P. Learning with drift detection. In: . [S.l.: s.n.], 2004. v. 8, p. 286–295.

GAMA, J.; SEBASTIÃO, R.; RODRIGUES, P. P. On evaluating stream learning algorithms. *Machine Learning*, v. 90, n. 3, p. 317–346, mar. 2013. ISSN 08856125.

GAMA, J.; ZLIOBAITE, I.; BIFET, A.; PECHENIZKIY, M.; BOUCHACHIA, A. *A survey on concept drift adaptation.* [S.l.]: Association for Computing Machinery, 2014.

GARCIA-MARTIN, E.; LAVESSON, N.; GRAHN, H. Identification of energy hotspots: A case study of the very fast decision tree. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. [S.l.]: Springer Verlag, 2017. v. 10232 LNCS, p. 267–281. ISBN 9783319571850. ISSN 16113349.

GARCÍA-MARTÍN, E.; LAVESSON, N.; GRAHN, H.; CASALICCHIO, E.; BOEVA, V. How to Measure Energy Consumption in Machine Learning Algorithms. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. [S.l.]: Springer Verlag, 2019. v. 11329 LNAI, p. 243–255. ISBN 9783030134525. ISSN 16113349.

GARCÍA-MARTÍN, E.; RODRIGUES, C. F.; RILEY, G.; GRAHN, H. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, Academic Press Inc., v. 134, p. 75–88, dez. 2019. ISSN 07437315.

GARCíA-MARTíN, E.; BIFET, A.; LAVESSON, N. Energy modeling of hoeffding tree ensembles. *Intelligent Data Analysis*, 2020.

GARCíA-MARTíN, E.; BIFET, A.; LAVESSON, N. Green accelerated hoeffding tree. <http://urn.kb.se/resolve?urn=urn:nbn:se:bth-19152>. 2020.

GARCíA-MARTíN, E.; LAVESSON, N.; GRAHN, H.; CASALICCHIO, E.; BOEVA, V. Energy-aware very fast decision tree. *Journal of Data Science and Analytics*, 2020.

GEMAQUE, R.; COSTA, A. F. J.; GIUSTI, R.; SANTOS, E. An overview of unsupervised drift detection methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 07 2020.

GIL, A. C. *Como elaborar projetos de pesquisa*. 4. ed.. ed. São Paulo, SP: Atlas, 2007. ISBN 978-85-224-3169-4.

GNU. *Processor And CPU Time (The GNU C Library)*. 2020. 1 p. Disponível em: <https://www.gnu.org/software/libc/manual/html{_}node/Processor-And-CPU-Time.html{\#}Processor-And-CP>.

GOMES, H. M.; BIFET, A.; READ, J.; BARDDAL, J. P.; ENEMBRECK, F.; PFHARINGER, B.; HOLMES, G.; ABDESSALEM, T. Adaptive random forests for evolving data stream classification. *Machine Learning*, v. 106, n. 9, p. 1469–1495, out. 2017. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/s10994-017-5642-8>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

HALEVY, A.; NORVIG, P.; PEREIRA, F. The unreasonable effectiveness of data. *Intelligent Systems, IEEE*, v. 24, p. 8 – 12, 05 2009.

HAN, J.; KAMBER, M.; PEI, J. *Data Mining. Concepts and Techniques, 3rd Edition (The Morgan Kaufmann Series in Data Management Systems)*. [S.l.: s.n.], 2011.

HARIRI, N.; MOBASHER, B.; BURKE, R. Adapting to user preference changes in interactive recommendation. In: *Proceedings of the 24th International Conference on Artificial Intelligence*. [S.l.]: AAAI Press, 2015. (IJCAI'15), p. 4268–4274. ISBN 9781577357384.

HOEFFDING, W. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, Informa UK Limited, v. 58, n. 301, p. 13–30, mar. 1963. Disponível em: <http://dx.doi.org/10.1080/01621459.1963.10500830>.

HU, H.; KANTARDZIC, M.; SETHI, T. S. *No Free Lunch Theorem for concept drift detection in streaming data classification: A review*. Wiley-Blackwell, 2020. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1327>.

INTEL. *Running Average Power Limit – RAPL | 01.org*. 2014. Disponível em: <https://01.org/blogs/2014/running-average-power-limit-\T1\textendash-rapl>.

JACOB, B.; NG, S. W.; WANG, D. T. Overview: On memory systems and their design. In: JACOB, B.; NG, S. W.; WANG, D. T. (Ed.). *Memory Systems*. San Francisco: Morgan Kaufmann, 2008. p. 1–54. ISBN 978-0-12-379751-3. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780123797513500023>.

KARAX, J. A. P.; MALUCELLI, A.; BARDDAL, J. P. Decision tree-based feature ranking in concept drifting data streams. In: *Proceedings of the ACM Symposium on Applied Computing*. [S.l.]: Association for Computing Machinery, 2019. Part F1477, p. 590–592. ISBN 9781450359337.

KASS, G. V. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, JSTOR, v. 29, n. 2, p. 119, 1980. Disponível em: <http://dx.doi.org/10.2307/2986296>.

KEOGH, E. Instance-based learning. In: _____. *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2010. p. 549–550. ISBN 978-0-387-30164-8. Disponível em: <https://doi.org/10.1007/978-0-387-30164-8_409>.

KHAMASSI, I.; SAYED-MOUCHAWEH, M.; HAMMAMI, ·. M.; GHÉDIRA, ·. K. Discussion and review on evolving data streams and concept drift adapting. *Evolving Systems*, v. 9, p. 1–23, 2018.

LAPLANTE, P. et al. *Comprehensive Dictionary of Electrical Engineering*. 1. ed. CRC Press, 2018. 454 p. Disponível em: <http://www.taylorandfrancis.com>.

LU, J.; LIU, A.; DONG, F.; GU, F.; GAMA, J.; ZHANG, G. Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering*, IEEE Computer Society, v. 31, n. 12, p. 2346–2363, abr. 2020. Disponível em: <http://arxiv.org/abs/2004.05785http://dx.doi.org/10.1109/TKDE.2018.2876857>.

MAHAJAN, D.; GIRSHICK, R.; RAMANATHAN, V.; HE, K.; PALURI, M.; LI, Y.; BHARAMBE, A.; MAATEN, L. van der. Exploring the Limits of Weakly Supervised Pretraining. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, 2018. v. 11206 LNCS, p. 185–201. ISBN 9783030012151. ISSN 16113349. Disponível em: <http://arxiv.org/abs/1805.00932>.

MARGARA, A.; RABL, T. Definition of Data Streams. In: *Encyclopedia of Big Data Technologies*. Springer International Publishing, 2019. p. 648–652. Disponível em: <https://link.springer.com/referenceworkentry/10.1007/978-3-319-77525-8{_}>.

MARTíN, E. G.; BIFET, A.; LAVESSON, N. Energy modeling of hoeffding tree ensembles. In: *Intelligent Data Analysis*. [s.n.], 2020. Disponível em: <https://www.acemap.info/paper/3731357>.

MARTÍN, E. G.; LAVESSON, N.; GRAHN, H. Energy efficiency in data stream mining. In: *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2015*. [S.l.]: Association for Computing Machinery, Inc, 2015. p. 1125–1132. ISBN 9781450338547.

MARTíN, E. G. *Energy Efficiency in Machine Learning : Approaches to Sustainable Data Stream Mining*. 267 p. Tese (Doutorado) — , Department of Computer Science, 2020.

MEHMOOD, E.; ANEES, T. Challenges and solutions for processing real-time big data stream: A systematic literature review. *IEEE Access*, v. 8, p. 119123–119143, 2020.

MICHAEL, M.; MOREIRA, J. E.; SHILOACH, D.; WISNIEWSKI, R. W. Scale-up x scale-out: A case study using nutch/Lucene. In: *Proceedings - 21st International Parallel and Distributed Processing Symposium, IPDPS 2007; Abstracts and CD-ROM*. [S.l.: s.n.], 2007. ISBN 1424409101.

MISHRA, A.; YAZICI, A.; MISHRA, D. Green information technology/ information system education: Curriculum views. *Technics Technologies Education Management*, v. 7, p. 679–686, 06 2012.

MONSOON. *High Voltage Power Monitor | Monsoon Solutions | Bellevue*. 2020. Disponível em: <https://www.msoon.com/high-voltage-power-monitor>.

MONTIEL, J.; READ, J.; BIFET, A.; KEGL, B. *Scikit-Multiflow: A Multi-output Streaming Framework*. [S.l.], 2018. v. 19, 1–5 p. Disponível em: <http://jmlr.org/papers/v19/18-251.html.>

MOTOROLA. *Chapter 2. PowerPC Register Set Chapter 2 PowerPC Register Set*. 2020. Accessed: 2020-10-10. Disponível em: <http://www.csit-sun.pub.ro/~cpop/Documentatie_SMP/Motorola_PowerPC/PowerPc/GenInfo/pemch2.pdf>.

Moulet, M.; Kodratoff, Y. From machine learning towards knowledge discovery in databases. In: *IEE Colloquium on Knowledge Discovery in Databases (Digest No. 1995/021 (A))*. [S.l.: s.n.], 1995. p. 5/1–5/3. ISSN null.

MOUSS, H.; MOUSS, D.; MOUSS, N.; SEFOUHI, L. Test of Page-Hinckley, an approach for fault detection in an agro-alimentary production system. In: *2004 5th Asian Control*

Conference. [s.n.], 2004. v. 2, p. 815–818. ISBN 0780388739. Disponível em: <https://www.researchgate.net/publication/4142016{_}Test{_}of{_}Page-Hinckley{_}an{_}approach{_}for{_}fault{_}detection{_}in{_}an{_}>.

MUKHERJEE, A.; MONDAL, S.; CHAKI, N.; KHATUA, S. *Naive Bayes and Decision Tree Classifier for Streaming Data Using HBase*. Singapore: Springer Singapore, 2019. 105–116 p. ISBN 978-981-13-3250-0. Disponível em: <https://doi.org/10.1007/978-981-13-3250-0_8>.

NAKANO, H. The efficiency of the arc lamp. *Transactions of the American Institute of Electrical Engineers*, v. 6, n. 5, p. 308–321, 1889. ISSN 00963860.

NOUREDDINE, A.; BOURDON, A.; ROUVOY, R.; SEINTURIER, L.; BOURDON, A. *A Preliminary Study of the Impact of Software Engineering on GreenIT*. [S.l.], 2012. 21–27 p. Disponível em: <https://hal.inria.fr/hal-00681560v3>.

NOUREDDINE, A.; BOURDON, A.; ROUVOY, R.; SEINTURIER, L. Runtime monitoring of software energy hotspots. In: *2012 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012 - Proceedings*. New York, New York, USA: ACM Press, 2012. p. 160–169. ISBN 9781450312042. Disponível em: <http://dl.acm.org/citation.cfm?doid=2351676.2351699>.

NOUREDDINE, A.; ROUVOY, R.; SEINTURIER, L. A review of energy measurement approaches. In: *Operating Systems Review (ACM)*. Association for Computing Machinery, 2013. v. 47, n. 3, p. 42–49. ISSN 01635980. Disponível em: <https://dl.acm.org/doi/10.1145/2553070.2553077>.

NOUREDDINE, A.; ROUVOY, R.; SEINTURIER, L. Unit testing of energy consumption of software libraries. In: *Proceedings of the ACM Symposium on Applied Computing*. New York, New York, USA: Association for Computing Machinery, 2014. p. 1200–1205. ISBN 9781450324694. Disponível em: <http://dl.acm.org/citation.cfm?doid=2554850.2554932>.

NOUREDDINE, A.; ROUVOY, R.; SEINTURIER, L. *Monitoring Energy Hotspots in Software Monitoring Energy Hotspots in Software Energy Profiling of Software Code*. [S.l.], 2015. v. 22, n. 3, 291–332 p. Disponível em: <http://www.eclipse.org/jetty>.

NTANOS, S.; ARABATZIS, G.; KONSTANTINOS, M.; PANAGIOTA, L.; CHALIKIAS, M. Energy consumption and co2 emissions on a global level. In: . [S.l.: s.n.], 2015.

OMONDI, A.; ATEYA, I. *Sustainable Energy Consumption in Data Centres*. 2019. Disponível em: <https://www.researchgate.net/publication/331821908_Sustainable_Energy_Consumption_in_Data_Centres>.

OZA, N. C. Online bagging and boosting. In: *2005 IEEE International Conference on Systems, Man and Cybernetics*. [S.l.: s.n.], 2005. v. 3, p. 2340–2345 Vol. 3.

PANIEGO, J. M.; GALLO, S.; Pi Puig, M.; CHICHIZOLA, F.; De Giusti, L.; BALLADINI, J. Analysis of RAPL energy prediction accuracy in a matrix multiplication application on shared memory. In: *Communications in Computer and Information Science*. [S.l.]: Springer Verlag, 2018. v. 790, p. 37–46. ISBN 9783319752136. ISSN 18650929.

PETROV, N. A.; PETROVA, N. N. Changes in the Energy of the World and North-East Asia. In: *2019 International Multi-Conference on Industrial Engineering and Modern Technologies, FarEastCon 2019*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2019. ISBN 9781728100616.

PHUNG, J.; LEE, Y. C.; ZOMAYA, A. Y. Modeling system-level power consumption profiles using RAPL. In: *NCA 2018 - 2018 IEEE 17th International Symposium on Network Computing and Applications*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2018. ISBN 9781538676592.

QUINLAN, J. R. Induction of decision trees. *Machine Learning*, v. 1, n. 1, p. 81–106, mar. 1986. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/BF00116251>.

RAO, K.; SUBRAMANI, S.; PRASAD, M.; SARAVANAN, A. Technical challenges and perspectives in batch and stream big data machine learning. *International Journal of Engineering & Technology*, v. 7, p. 48, 12 2017.

REINSEL, D.; GANTZ, J.; RYDNING, J. *The Digitization of the World From Edge to Core*. [S.l.], 2018.

RUDIO, F. V. *Introdução ao projeto de pesquisa cientifica*. 4. ed.. ed. Petrópolis: Vozes, 1980.

SALZBERG, S. L. C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, v. 16, n. 3, p. 235–240, set. 1994. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/BF00993309>.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, v. 3, n. 3, p. 210–229, jul. 1959. ISSN 0018-8646.

SCARFO, A. All silicon data center, the energy perspectives. In: *Proceedings - 27th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2013*. [S.l.: s.n.], 2013. p. 1617–1622. ISBN 9780769549521.

SCHWARTZ, R.; DODGE, J.; SMITH, N. A.; ETZIONI, O. Green AI. jul. 2019. Disponível em: <http://arxiv.org/abs/1907.10597>.

SHAIKH, E.; MOHIUDDIN, I.; ALUFAISAN, Y.; NAHVI, I. Apache spark: A big data processing engine. In: . [S.l.: s.n.], 2019. p. 1–6.

SILVER, D.; HUBERT, T.; SCHRITTWIESER, J.; ANTONOGLOU, I.; LAI, M.; GUEZ, A.; LANCTOT, M.; SIFRE, L.; KUMARAN, D.; GRAEPEL, T.; LILLICRAP, T.; SIMONYAN, K.; HASSABIS, D. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. dez. 2017. Disponível em: <http://arxiv.org/abs/1712.01815>.

SUN, C.; SHRIVASTAVA, A.; SINGH, S.; GUPTA, A. *Revisiting Unreasonable Effectiveness of Data in Deep Learning Era*. 2017.

TANTITHAMTHAVORN, C.; MCINTOSH, S.; HASSAN, A. E.; MATSUMOTO, K. An Empirical Comparison of Model Validation Techniques for Defect Prediction Models. *IEEE Transactions on Software Engineering*, Institute of Electrical and Electronics Engineers Inc., v. 43, n. 1, p. 1–18, jan. 2017. ISSN 00985589.

THORAT, C. G.; INAMDAR, V. S. Energy Measurement of Encryption Techniques Using RAPL. In: *2017 International Conference on Computing, Communication, Control and Automation, ICCUBEA 2017*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2018. ISBN 9781538640081.

VASEEKARAN, G. *Big Data Battle : Batch Processing vs Stream Processing*. 2017. <https://medium.com/@gowthamy/big-data-battle-batch-processing-vs-stream-processing-5d94600d8103>. (Accessed on 10/17/2020).

VENKATESH, A.; KANDALLA, K.; PANDA, D. K. Evaluation of energy characteristics of MPI communication primitives with RAPL. In: *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, IPDPSW 2013*. [S.l.]: IEEE Computer Society, 2013. p. 938–945. ISBN 9780769549798.

WAH, C.; BRANSON, S.; WELINDER, P.; PERONA, P.; BELONGIE, S. *The Caltech-UCSD Birds-200-2011 Dataset*. [S.l.], 2011.

WANG, C.; WANG, Y. Discovering consumer's behavior changes based on purchase sequences. In: *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*. [S.l.: s.n.], 2012. p. 642–645.

WEBB, G. I. Naïve bayes. In: _____. *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2010. p. 713–714. ISBN 978-0-387-30164-8. Disponível em: <https://doi.org/10.1007/978-0-387-30164-8_576>.

XIAO, j. *Beer and Nappies*. 2014. Disponível em: <https://bigdatabigworld.wordpress.com/2014/11/25/beer-and-nappies/>.

XU, S.; WANG, J. Dynamic extreme learning machine for data stream classification. *Neurocomputing*, Elsevier B.V., v. 238, p. 433–449, may 2017. ISSN 18728286.

ZHANG, X.; WANG, Z.; GLOY, N.; CHEN, J. B.; SMITH, M. D. System support for automatic profiling and optimization. In: *Proceedings of the sixteenth ACM symposium on Operating systems principles - SOSP '97*. New York, New York, USA: Association for Computing Machinery (ACM), 1997. p. 15–26. Disponível em: <http://portal.acm.org/citation.cfm?doid=268998.266640>.

ZHANG, Y.; CHU, G.; LI, P.; HU, X.; WU, X. Three-layer concept drifting detection in text data streams. *Neurocomputing*, Elsevier B.V., v. 260, p. 393–403, oct 2017. ISSN 18728286.

ZHENG, X.; LI, P.; CHU, Z.; HU, X. A survey on multi-label data stream classification. *IEEE Access*, v. 8, p. 1249–1275, 2020.