Evaluating the Four-Way Performance Trade-Off for Data Stream Classification in Edge Computing

Jessica Fernandes Lopes, Everton Jose Santana, Victor G. Turrisi da Costa, Bruno Bogaz Zarpelão, and Sylvio Barbon Junior

***Abstract*—Edge computing (EC) is a promising technology capable of bridging the gap between Cloud computing services and the demands of emerging technologies such as the Internet of Things (IoT). Most EC-based solutions, from wearable devices to smart cities architectures, benefit from Machine Learning (ML) methods to perform various tasks, such as classification. In these cases, ML solutions need to deal efficiently with a huge amount of data, while balancing predictive performance, memory and time costs, and energy consumption. The fact that these data usually come in the form of a continuous and evolving data stream makes the scenario even more challenging. Many algorithms have been proposed to cope with data stream classification, e.g., Very Fast Decision Tree (VFDT) and Strict VFDT (SVFDT). Recently, Online Local Boosting (OLBoost) has also been introduced to improve predictive performance without modifying the underlying structure of the decision tree produced by these algorithms. In this work, we compared the four-way relationship among time efficiency, energy consumption, predictive performance, and memory costs, tuning the hyperparameters of VFDT and the two versions of SVFDT with and without OLBoost. Experiments over 6 benchmark datasets using an EC device revealed that VFDT and SVFDT-I were the most energy-friendly algorithms, with SVFDT-I also significantly reducing memory consumption. OLBoost, as expected, improved the predictive performance, but caused a deterioration in memory and energy consumption.**

***Index Terms*—Machine learning, data stream mining, energy efficiency, Internet of Things, edge computing.**

I. Introduction

THE GROWING adoption of Internet of Things (IoT) solutions demands methods capable of extracting accurate information from raw sensor data [1], [2] and managing huge volumes of data [3]. Over the years, different problems involving data analysis have been tackled successfully with Machine Learning (ML). For example, performing anomaly detection for cybersecurity, spam filtering, weather forecasting, medical diagnosis, image classification and segmentation, and driving autonomous vehicles. However, IoT systems have some particularities that pose additional challenges towards ML algorithms application.

Smart grids are a typical example. They make use of a large number of sensors for monitoring customers' energy consumption, electric quantities from power grid devices, and data from the surrounding environment (e.g., temperature and humidity) [4]. These data are continuously collected from multiple points, which are spread across large geographical areas. Then, ML algorithms might be applied to build intelligent solutions using this data. However, most sensors are resource-constrained and cannot run ML algorithms, while sending all this data to a remote dedicated server may cause latency and bandwidth issues.

To avoid the strain of transmitting all data to a remote data center, micro servers can be placed at the network edge, close to the sensors. These micro servers are responsible for processing the collected data and making decisions when possible, reducing the dependence on central servers, which can improve the overall performance of the entire system. Deploying devices to allow computation at the network edge, close to the data source, is the core idea of the Edge Computing (EC) paradigm [5], [6].

Even though EC devices seem to solve the problem of running ML algorithms over sensor data completely, there are some challenges yet to overcome. Edge devices are certainly more robust than IoT sensors, but they still may struggle to meet the requirements of some ML algorithms since they are not high-end servers. Alongside EC limitations, sensor data is also an issue for traditional ML. Firstly, it usually comes in the form of a continuous and evolving data stream, which is potentially infinite, meaning that storing data is very difficult [7]–[9]. Additionally, the evolving nature of data streams makes ML solutions need to continuously update their learning models to produce fast and reliable results.

Many ML algorithms have been proposed for data stream classification [8], [10]–[13]. Among them, the Very Fast Decision Tree (VFDT) [10] induces a decision tree in an online fashion. A modification of the VFDT, called Strict VFDT (SVFDT) [13], was also proposed to control tree growth while maintaining predictive performance and processing time. Furthermore, Online Local Boosting (OLBoost) [14] was introduced to increase the predictive performance of online decision trees without modifying their resulting decision tree.

Manuscript received August 30, 2019; revised January 16, 2020; accepted March 24, 2020. Date of publication March 30, 2020; date of current version June 10, 2020. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES)-Finance Code 001, the National Council for Scientific and Technological Development-Brazil (CNPq)-Grant of Project 420562/2018-4 and Paraná State Agency Fundação Araucária. The associate editor coordinating the review of this article and approving it for publication was R. Pasquini. *(Corresponding author: Everton Jose Santana.)*

Jessica Fernandes Lopes, Everton Jose Santana, Bruno Bogaz Zarpelão, and Sylvio Barbon Junior, are with the Department of Computer Science, State University of Londrina, Paraná 86057-970, Brazil (e-mail: jessicafernandes@uel.br; evertonsantana@uel.br; brunozarpelao@uel.br; barbon@uel.br).

Victor G. Turrisi da Costa is with the Department of Information Engineering and Computer Science, University of Trento, 38123 Trento, Italy (e-mail: vg.turrisidacosta@unitn.it).

Although they were vastly evaluated in multiple datasets [7], [8], [10]–[15], energy costs were only assessed along with other performance measurements for VFDT and SVFDT in our previous work [16]. However, the evaluation of these algorithms in an EC device was not addressed. Furthermore, solutions in this scenario need to be even more aware of their computational costs.

In this work, we consider that ML algorithms for stream classification may be suitable for EC environments. Therefore, we evaluate the trade-off among four key performance aspects of the execution of data stream classification in an EC device, namely energy consumption, predictive performance, memory costs, and time costs. The experiments were carried out for VFDT and SVFDT with and without OLBoost. VFDT acts as a default baseline for our comparisons, while SVFDT is a more memory friendly alternative. OLBoost provides a way to increase predictive performance without seriously compromising energy, memory, and time costs, as ensemble solutions would. The tests were performed over six binary benchmark datasets in a Raspberry Pi, which has been frequently pointed as a platform for EC devices [6], [17]–[19].

This work is an extension of [16], whose main goal was to analyze the four-way relationship of predictive performance, memory and time costs and energy consumption of VFDT and SVFDTs, without OLBoost, in a desktop computer. That work was an introductory study and its result motivated further analysis in a platform actually employed in EC applications. In this sense, we extend the previous work in the following main aspects:

1) To address the EC paradigm, experiments were performed in a Raspberry Pi, a typical EC device with limited memory and processing capability;
2) The impact of OLBoost in the VFDT and SVFDTs was also assessed.

The remainder of this work is organized as follows. Section II highlights the importance of considering energy, accuracy, memory and time in the EC context. Section III describes the data stream mining algorithms used in this work. In Section IV we present the experimental setup. Results and Discussions are contained in Section V. Section VI presents the main conclusions and directions of future work.

II. EC Performance Trade-Off

When selecting the best algorithm for a particular application, different aspects must be considered. For EC, it is important to consider the adequacy of an algorithm in the following main aspects: energy consumption, predictive performance, memory resources and time costs. These aspects are essential to address the device limitations and the evolving nature of collected data.

Time costs: Traditional ML algorithms have two separated time costs: one of inducing the model and another of performing predictions [20]. However, when using data stream algorithms, both times are combined into a single unit since these algorithms need to be continuously updated while also performing predictions when requested [9], [13]. In other words, using traditional ML algorithms consists of first inducing the algorithm using data collected during some time span and then performing predictions. On the other hand, both phases occur in parallel in data stream algorithms. In most IoT and EC cases, the longer the time to induce the model, the more accurate it becomes [21]. However, as inducing time increases, energy consumption can also increase, meaning that time is an influential factor.

Energy consumption: IoT applications and EC devices, two common domains for data stream mining, may rely on batteries [6]. This highlights the importance of algorithms with low power consumption because they can contribute to reducing the need for replacing, recharging or disposing of those batteries [22], [23]. Also, energy-efficient computing is fundamental since information and communications technology has been pointed out as responsible for significant power demand, which generates carbon and other pollutant emissions during its production [24], [25]. Lastly, the greater the energy consumption, the greater the heat produced, which affects the durability and reliability of hardware metrics [22] and demands higher expenses with cooling systems [26].

Predictive performance: Several EC-based applications depend on acquiring data for online analysis and monitoring. The adoption of classifiers in this type of system may seek different objectives such as detecting attacks [27] or identifying faults in the system [28]. Consequently, the required predictive performance is highly dependent on the application's nature. A very critical system, for example, may require higher predictive performance, even if this means to use more computational resources. However, in EC devices, those resources are limited. For this reason, it is important to investigate how predictive performance is affected by strategies adopted by data stream algorithms to cope with resource limitations. Furthermore, the volume of data and their fast-changing behavior have revealed a challenging task [29].

Memory resources: Unlike most data centers and cloud platforms, which rely on dedicated high-end servers, EC setups may consist of devices like single board computers, routers, gateways, and access points [17], [18]. Most of these devices are not primarily designed to run intensive computing tasks, so they have significantly less memory than typical servers. This issue emphasizes the need for algorithms that are lightweight and use limited computational resources. One class of algorithms capable of dealing with these constraints are data stream algorithms. Memory management is even more relevant for non-parametric algorithms because the number of free parameters evolves with the number of training examples [30]. For other types of algorithms, like linear models, memory costs depend on the number of features of the problem. Modifications of the VFDT provide more memory-friendly models due to the capability of online updates, without the need to store data. Although this leads to lower predictive performance, the amount of memory used decreases and the accuracy is still satisfactory [13]. Since computational resources are limited in EC devices, the trade-off of achieving a slightly lower accuracy at the cost of significantly saving memory is very beneficial.

III. Data Stream Classification

ML algorithms, traditionally, model knowledge from static and previously stored datasets [13]. However, there is a

growing demand for solutions capable of dealing with huge volumes of data that come in the form of streams. This assumes that new data can arrive at any time and storing these data is impracticable. It is important to highlight that learning from a data stream means to create and keep the model updated throughout the stream. Additionally, tackling concept drifts, i.e., the change of data distribution over time, is also important [7], [31]. Moreover, in EC scenarios, model updates must be fast, and the memory available and processing power are limited, requiring suitable algorithms capable of efficiently managing processing time and memory space while keeping competitive accuracy.

Among several data stream classification algorithms, the original extension of decision trees for dealing with data streams, VFDT, is one of the most widely used and subject to modifications [32]. The SVFDT obtained similar accuracy results to the VFDT for several problems [13] while expending less computational resources, which meets the requirements of EC devices. Recently, OLBoost [14] improved the predictive performance of both algorithms without increasing resource consumption in the way an ensemble solution would. Next, the three algorithms are further detailed.

A. Very Fast Decision Tree

Hoeffding Bound (HB) theorem has been the foundation for many of data stream algorithms, e.g., VFDT [10]. VFDT is a tree-based ML algorithm which employs HB to perform splits when growing a tree. Its execution is based on a continuous variable *v*, whose values are bounded by the interval $[v_{min}, v_{max}]$, with a range of values $R = v_{max} - v_{min}$, that is independently observed *n* times and, according to these observations, has a mean of $\overline{v}$. Then, the HB theorem states that the mean of this variable when $n \to \infty$ is $\overline{v_{n\to\infty}}$ and bounded by the interval $[\overline{v} - \epsilon, \overline{v} + \epsilon]$ with statistical probability $1 - \delta$, where

$$\epsilon = \sqrt{\frac{R^2\, ln\left(\frac{1}{\delta}\right)}{2\, n}}. \tag{1}$$

VFDT uses the HB theorem to check whether the best split candidate would remain the best if the tree received additional instances. It is important to note the split attempt is based on a heuristic measure *G*(.), such as Information Gain (IG) or Gini Index (GI), by computing ϵ and checking if $G(best) - G(second\_best) = \Delta G < \epsilon$, where *G*(*best*) and *G*(*second_best*) are the *G*(.) values of the best split candidate feature and the second best one.

A model is created by VFDT using a single instance at a time, requiring limited computational memory resources. Under realistic assumptions, it has the same asymptotic performance as the induction of a decision tree induced by a standard batch algorithm [9]. The algorithm maintains and updates the instances' class distribution for each leaf to compute the number of instances of each class. Moreover, computing numerical estimators are also employed to keep the relationship between feature values and class distributions.

To increase the predictive performance, lightweight classification algorithms are used at the leaves, with Adaptive Naive Bayes (ANB) being the most widely applied [33]. Also, three hyperparameters (*GP*, τ and δ) need to be set to enhance the performance. *GP* states the amount of instances between each split attempt. A high value will result in fewer split attempts and a smaller tree. On the other hand, a low *GP* allows a faster adaptation, resulting in a larger tree. τ allows splits when a high amount of instances were observed regardless of HB was satisfied. When increasing it, fewer instances will be needed to ignore HB. An extremely high value coupled with a low *GP* results in an overfitted and low-performance tree. On the other way round, very low τ values may retain tree growth, resulting in a small and low-performance tree. Finally, δ affects the computation of HB. Considering higher values, ϵ will be smaller, which has a similar effect as increasing τ, whereas a small δ results in a larger ϵ and produces the same effect as decreasing τ.

Although the *GP* and τ hyperparameters influence in predictive performance, memory and time costs have already been evaluated in [10], [13], [15], [25], [34], [35], energy consumption has, to the best of our knowledge, only been considered for this algorithm in [25] and [16]. A more in-depth analysis of the relationship of the hyperparameters and usage of these resources in EC devices is needed.

B. Strict Very Fast Decision Tree

The SVFDT algorithm is a modification of VFDT [13], which creates smaller decision trees than VFDT with very similar predictive performance. It implements additional rules to block tree growth using the following φ function:

$$\varphi(x, X) = \begin{cases} \text{True}, & \text{if } x \geq \overline{X} - \sigma(X) \\ \text{False}, & \text{otherwise} \end{cases} \tag{2}$$

where *X* is a set of observed values, $\overline{X}$ is their mean, $\sigma(X)$ is their standard deviation, and *x* is a new observation.

There are two SVFDT algorithm versions: the SVFDT-I and SVFDT-II, both using three assumptions: i) A leaf node should only split when there is a minimum uncertainty of class assumption (e.g., high entropy) associated with the instances, according to previous and current statistics. ii) A similar amount of instances should be observed across all leaf nodes. iii) The feature used for splitting should significantly decrease uncertainty (e.g., a high IG) according to previous statistics. The difference between the versions is related to the fact that SVFDT-II employs additional skipping mechanisms to speed-up growth when class uncertainty is too high or this uncertainty is largely reduced.

SVFDT, as evaluated in [13] and [15], reduces memory consumption without compromising predictive performance and achieves time speed-up. Although the computations added by SVFDT may be insignificant when considering the mentioned performance metrics, energy consumption can be affected, positively or negatively, as shown in [16].

C. Online Local Boosting

OLBoost is an algorithm created to increase predictive performance, which works alongside online decision trees (ODTs) [14]. It uses the assumption that instances being

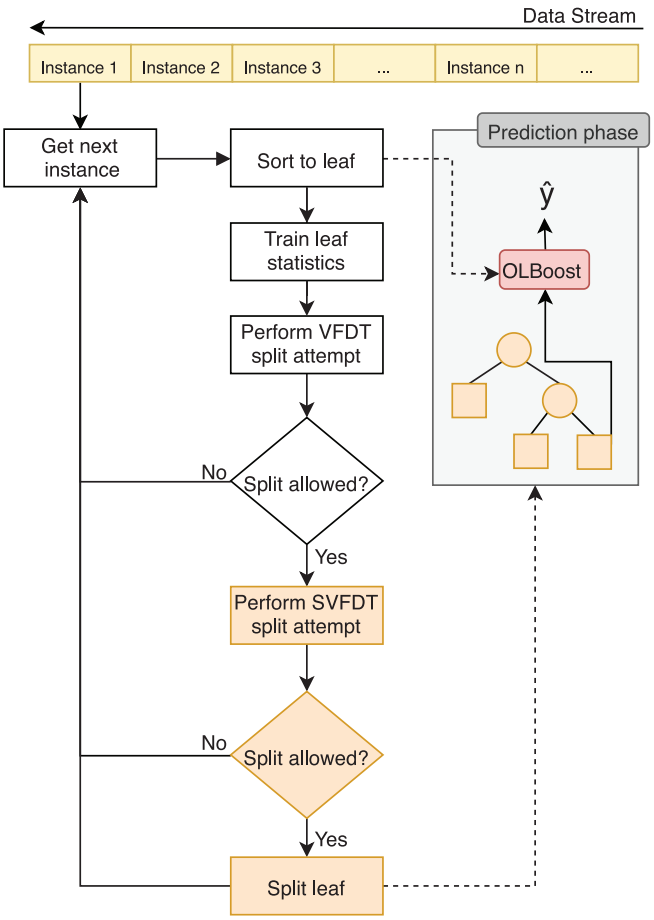


Fig. 1. OLBoost coupled with VFDT/SVFDT in the training and prediction phases.

wrongly classified are required to be used more times when inducing a model, while, on the other hand, instances that are easily classified can even be ignored during the training phase. Note that it works using an ANB or a simple majority class predictor. Figure 1 shows how OLBoost works when coupled with VFDT and the SVFDTs.

OLBoost is grounded on the presumption that the instances of an (unbounded) data stream are presented one at a time to the online learning algorithm. After the ODT algorithm processes an instance, OLBoost outputs the probabilities vector of this instance's class and extracts sampling weights from a Poisson distribution, similarly to [36]. Thus, the predictor (NB, ANB, or majority class) used by OLBoost is updated based on the probability estimated by the leaf of an instance being from its real class. Subsequently, the instance is used to update the leaf statistics with a normal weight of 1 and a split attempt is performed.

IV. Materials and Methods

This section presents the datasets employed in this work, the evaluation metrics used and the experimental setup.

A. Datasets

To compare VFDT and the SVFDTs energetic efficiency, predictive performance, and memory and time costs, we used 6 binary benchmark datasets for data stream classification. These datasets cover both real-life and synthetic datasets. Sea and usenet datasets present, in addition, concept drifts. Table I summarizes the main aspects of each dataset.

A brief explanation of the datasets follows:

- Agrawal dataset [37], [41]: generates data according to classification functions using six numeric (salary, commission, age, hvalue, hyears, loan) and three categorical (elevel, car, zipcode) features. This dataset was generated using the default settings in the MOA framework [38] (function 1 and perturbation fraction of 0.05).
- Airline dataset [38]: each instance of this dataset corresponds to a flight from one airport to another. The instances have 2 class values, which describe if that flight was delayed or not, given the information of the scheduled departure.
- Hyperplane dataset (hyper) [38]: this dataset originated from the rotation in a hyperplane configured to generate two classes based on 10 numeric features.
- Electricity Pricing dataset (elec) [38]: this dataset was collected from the Australian New South Wales Electricity Market, where electricity prices are not fixed. The prices are affected by the demand and supply of the market and updated every five minutes. The dataset output identifies the changes in the price (2 possible classes: up or down) relative to a moving average of the last 24 hours. Additionally, this dataset exhibits temporal dependencies among instances.
- SEA dataset (sea) [39]: all features of this dataset are numeric between 0 and 10, with only the first two being relevant. An instance is considered as being from class 0 if relevant_feature1 + relevant_feature2 > θ. This threshold θ changes four times every 15,000 instances with values 8, 9, 7, and 9.5 resulting in a total of 60,000 instances. Additionally, the dataset has 10% of noise.
- Usenet dataset (usenet) [40]: simulation of news filtering with concept drift related to the change of interest of a user over time. Each instance has 659 binary features describing the presence or absence of a respective word in the piece of text. The two classes indicate if a virtual user is interested in this kind of news or not. The concept drift occurs by making this virtual user lose interest in some types of news and gain in others.

B. Energy Consumption Measurement

In computing systems, energy, i.e., the capacity for producing work, is delivered as electricity [26]. To address the task of measuring it, some consumption monitors were developed without the need for additional hardware metrics [42], [43]. In this work, we adopted PowerAPI, an application programming interface (API) that monitors the power usage of running processes based on information collected from hardware elements of the operating system [43]. When compared with a power meter, the PowerAPI margin of error is between 0.5% and 3% [44], motivating choosing it in this work.

TABLE I
SUMMARY OF THE DATASETS USED IN THE EXPERIMENT

Dataset	# instances	# features			# classes	% majority class
		numeric	binary	categorical		
agrawal [37]	1,000,000	6	0	3	2	0.672
airlines [38]	539,383	3	0	4	2	0.555
elec [38]	45,312	6	0	1	2	0.575
hyper [38]	250,000	10	0	0	2	0.500
sea [39]	60,000	3	0	0	2	0.627
usenet [40]	5,930	0	658	0	2	0.504

Generally, energy consumption (E) is computed as the product of power (P) and time (t),

$$E = P \times t. \tag{3}$$

Usually, power changes over time, so that the total used energy for performing a task is given by

$$E = \int_{t_0}^{t_F} P(t) \times dt. \tag{4}$$

where t_0 means the beggining of the task, and t_F, the end.

Assuming that the power usage is sampled at fixed and sufficiently small time intervals t_S, Equation 4 can be approximated to

$$E = \sum_{i=1}^{N} P_i \times t_S, \tag{5}$$

where N represents the total number of samples.

Specifically for our case, based on the computational costs (P_{comp}) calculated by PowerAPI software approach, it is possible to compute the total energy consumption of a process following:

$$E[J] = \sum_{i=1}^{N} P_{comp,i}[W] \times t_S[s]. \tag{6}$$

Although energy consumption and processing time are related, measuring energy consumption is still important because power introduces a non-linearity to this relationship. For instance, reducing the frequency of a processor for specific processes can lead to longer executions along with a decrease in power. This can reduce energy consumption even though the time increases [45].

Physically, energy-efficiency is the ratio between the work done and the total energy spent to accomplish this work, but as expressed in [46], it is a generic term and can be understood as the ratio between the useful output of a process and the energy input into a process. In this sense, when comparing the energy-efficiency of two or more algorithms, the most efficient is the one that uses the least energy to produce the same amount of useful output.

C. Evaluation Setup

VFDT, SVFDT-I, SVFDT-II, and their versions with OLBoost were chosen as learning algorithms. The conceptual simplicity and extensive use of trees in streams [47] motivated the choice of this family of algorithms. We evaluated the algorithms while varying two hyperparameters (GP and τ). These settings are presented in Table II. Additionally, we used *H* and *IG* as impurity and gain metrics.

TABLE II
HYPERPARAMETERS FOR THE BASE LEARNERS USED IN THE EXPERIMENT

Hyperparameter	Setting
GP	(500, 750, 1000)
τ	(0.05, 0.10)
δ	10^{-5}
Numeric Estimator	Gaussian - 100 bins
Leaf Predictor	ANB

For each dataset, the prequential learning method [9] was performed four times in a row for each algorithm to compute the mean energy consumption, accuracy, processing time and memory consumption. This learning method consists of presenting each instance individually to the algorithm, asking for a classification (which is used to evaluate the model, e.g., compute its accuracy) and then using the instance to update the model. We did not use the results of the first of the four executions due to its unstable behavior. In the first execution of each pair classifier-dataset, the needed data was accessed for the first time after the device was turned on. Afterward, for the remaining executions, the operating system optimized the access to this data due to its recurrent use. As a result, the performance metrics collected for the first execution are unstable, while the metrics collected for the other executions are more stable. Along with accuracy, Kappa M [48] was also used to measure how a classifier performs in comparison with a majority class predictor.

The algorithms and experimental environment were implemented in Python and Cython and the code is publicly available.[1] The experiments were performed on a Raspberry Pi 2 Model B with a thermal design power (TDP) of 4 W. It has 900 MHz quad-core ARM Cortex A7 CPU. However, the code uses only a single processor. Raspberry Pi is a single-board computer widely used in EC projects, as can be seen in [6], [17]–[19]. The PowerAPI was configured to capture the mean power of the intervals, using the *procfs* module [49] with a sampling period of 1 ms.

In order to compare multiple algorithms across multiple datasets, we employed the Friedman's statistical test with a significance level (α) of 0.05. After that, we applied post-hoc test of Nemenyi, which generates a Critical Distance (CD) and

[1] https://github.com/vturrisi/pystream

TABLE III
MEAN AND STANDARD DEVIATION VALUES FOR ENERGY CONSUMPTION, ACCURACY, KAPPA M, MEMORY SIZE AND TIME FOR EACH COMBINATION OF ALGORITHM AND DATASET. THE BOLD VALUES CORRESPOND TO THE BEST MEAN RESULT FOR EACH METRIC IN THE DATASET

Dataset	OLBoost	Algorithm	Time (s)	Energy (J)	Accuracy	Kappa M	Memory (MB)
agrawal	✓	SVFDT-I	2028.584 ± 51.863	76.790 ± 7.509	0.946 ± 0.008	0.834 ± 0.026	0.349 ± 0.134
		SVFDT-II	2030.648 ± 59.915	76.579 ± 7.545	**0.948 ± 0.003**	0.841 ± 0.008	3.328 ± 1.507
		VFDT	2024.426 ± 50.394	73.500 ± 9.090	**0.948 ± 0.001**	0.841 ± 0.003	9.519 ± 5.811
	-	SVFDT-I	1749.274 ± 53.409	72.929 ± 8.636	0.945 ± 0.008	0.834 ± 0.026	**0.194 ± 0.073**
		SVFDT-II	1744.842 ± 54.420	79.409 ± 9.209	**0.948 ± 0.003**	0.840 ± 0.009	1.829 ± 0.823
		VFDT	**1719.297 ± 31.572**	**69.355 ± 9.136**	**0.948 ± 0.001**	**0.842 ± 0.003**	5.280 ± 3.203
airlines	✓	SVFDT-I	2401.612 ± 233.567	77.643 ± 7.894	**0.664 ± 0.001**	**0.245 ± 0.003**	12.737 ± 2.0230
		SVFDT-II	2032.910 ± 502.548	77.693 ± 9.121	0.660 ± 0.001	0.237 ± 0.003	25.281 ± 7.484
		VFDT	1609.429 ± 200.906	78.633 ± 10.021	0.660 ± 0.004	0.237 ± 0.009	56.510 ± 27.661
	-	SVFDT-I	2249.651 ± 247.720	75.413 ± 6.663	0.655 ± 0.001	0.225 ± 0.002	**6.479 ± 1.029**
		SVFDT-II	1873.934 ± 511.831	77.513 ± 8.110	0.655 ± 0.001	0.225 ± 0.003	12.860 ± 3.807
		VFDT	**1438.165 ± 186.369**	**72.240 ± 8.617**	0.656 ± 0.002	0.228 ± 0.005	28.744 ± 14.069
elec	✓	SVFDT-I	442.696 ± 9.289	10.148 ± 0.375	0.823 ± 0.002	0.583 ± 0.004	0.299 ± 0.042
		SVFDT-II	452.049 ± 16.189	9.925 ± 0.312	0.825 ± 0.002	0.588 ± 0.006	0.371 ± 0.089
		VFDT	469.843 ± 21.225	9.577 ± 0.358	**0.832 ± 0.007**	**0.605 ± 0.017**	0.543 ± 0.204
	-	SVFDT-I	416.745 ± 17.256	8.549 ± 0.211	0.795 ± 0.001	0.517 ± 0.002	**0.175 ± 0.024**
		SVFDT-II	419.841 ± 12.917	**8.510 ± 0.287**	0.797 ± 0.003	0.521 ± 0.007	0.217 ± 0.053
		VFDT	**408.030 ± 10.307**	9.010 ± 0.544	0.797 ± 0.005	0.521 ± 0.014	0.319 ± 0.120
hyper	✓	SVFDT-I	957.673 ± 33.085	59.754 ± 6.575	**0.896 ± 0.001**	**0.793 ± 0.001**	0.436 ± 0.118
		SVFDT-II	966.582 ± 96.822	62.019 ± 8.890	0.891 ± 0.004	0.783 ± 0.009	0.885 ± 0.335
		VFDT	960.109 ± 54.292	61.840 ± 7.684	0.890 ± 0.008	0.779 ± 0.018	2.525 ± 1.272
	-	SVFDT-I	885.611 ± 28.961	59.288 ± 6.306	0.889 ± 0.001	0.778 ± 0.003	**0.257 ± 0.069**
		SVFDT-II	908.279 ± 47.456	**59.180 ± 7.562**	0.885 ± 0.003	0.769 ± 0.006	0.523 ± 0.196
		VFDT	**875.786 ± 27.518**	62.440 ± 6.347	0.886 ± 0.005	0.772 ± 0.012	1.486 ± 0.749
sea	✓	SVFDT-I	452.257 ± 12.585	10.199 ± 0.330	0.851 ± 0.001	0.601 ± 0.003	0.068 ± 0.017
		SVFDT-II	456.848 ± 12.176	10.132 ± 0.250	0.855 ± 0.002	0.612 ± 0.007	0.140 ± 0.089
		VFDT	462.230 ± 28.884	9.942 ± 0.551	**0.860 ± 0.003**	**0.625 ± 0.01**	0.264 ± 0.120
	-	SVFDT-I	420.431 ± 27.862	9.010 ± 0.386	0.848 ± 0.001	0.593 ± 0.002	**0.048 ± 0.012**
		SVFDT-II	421.327 ± 16.298	**8.930 ± 0.322**	0.849 ± 0.001	0.594 ± 0.004	0.099 ± 0.062
		VFDT	**403.327 ± 15.799**	9.148 ± 0.720	0.850 ± 0.002	0.597 ± 0.006	0.185 ± 0.084
usenet	✓	SVFDT-I	719.332 ± 16.093	46.929 ± 3.966	0.549 ± 0.007	0.091 ± 0.010	2.332 ± 0.340
		SVFDT-II	719.017 ± 33.423	49.469 ± 6.499	0.548 ± 0.007	0.089 ± 0.014	2.999 ± 1.163
		VFDT	723.689 ± 40.362	**45.283 ± 3.489**	0.550 ± 0.009	0.094 ± 0.019	3.886 ± 1.732
	-	SVFDT-I	643.392 ± 26.219	46.092 ± 4.947	0.551 ± 0.007	0.095 ± 0.015	**1.184 ± 0.173**
		SVFDT-II	641.478 ± 18.623	46.912 ± 6.721	0.552 ± 0.006	0.097 ± 0.013	1.522 ± 0.590
		VFDT	**636.138 ± 38.609**	46.201 ± 6.063	**0.554 ± 0.006**	**0.100 ± 0.013**	1.972 ± 0.879

can be illustrated in a diagram. The performance of algorithms connected by the CD are statistically equal at α [50].

V. RESULTS AND DISCUSSIONS

A. General Results

Table III presents the performance metrics for the datasets and algorithms. The mean and standard deviation values were computed based on the variation caused by the τ and *GP* hyperparameters.

From the table, some particular results can be observed for the different datasets: at the same time that agrawal was the dataset with the highest accuracy (with mean values varying from 0.945 to 0.948), it was one of most time costing, and energy-consuming dataset. For the same dataset, the mean energy consumption varied from 69.355 to 79.409 J, depending on the chosen algorithm. It is also noteworthy that the mean memory varied from 0.194 to 9.519 MB according to the algorithm, which is a significant difference. However, this was the only dataset that presented more than 2 best metrics for the same algorithm.

Airlines is a dataset similar to agrawal in number of features, with about half of instances. However, the energy and time spent were very comparable and this dataset required, in general, more memory than the previous one. For airlines, the mean memory varied from 6.479 to 56.510 MB.

The elec dataset presented lower sensitivity to algorithm selection than the previous datasets, i.e., the variation in energy, accuracy, memory and time when using different algorithms for elec was not as noticeable as for agrawal and airlines. Also, the standard deviations related to this dataset were one of the lowest among the experimented datasets, meaning that hyperparameter modification incurred in a modest change in performance.

The sea dataset registered results comparable to elec, with small mean and standard deviation. The biggest standard deviation values were found for the time metric.

The usenet was the only dataset which OLBoost did not deliver the best accuracy performance for. It also presented the lowest Kappa M, being very close to 0, meaning that the performance of the classifier was comparable to a majority baseline (i.e., a classifier that always outputs the majority class).

The general results of the table are: OLBoost tended to provide the best accuracy and Kappa M whereas the default versions of VFDT and SVFDT tended to generate lower costs in energy, memory and time.

In the following Sections (V-B–E), we will discuss each performance metric separately. To this end, we constructed

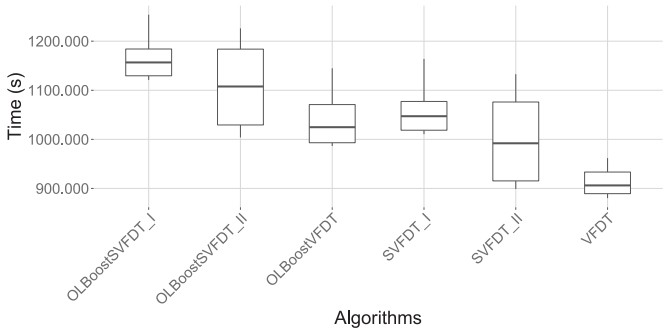


Fig. 2. Boxplots of time of VFDT, SVFDT-I and SVFDT-II, and their versions with OLBoost.

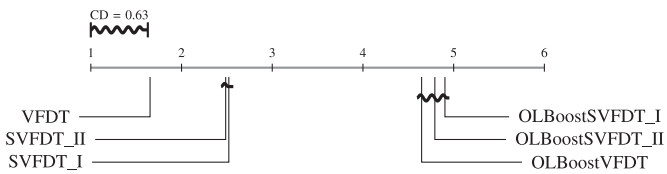


Fig. 3. Comparison of time performance among VFDT, SVFDT-I and SVFDT-II, and their versions with OLBoost according to Nemenyi test with $\alpha = 0.05$.

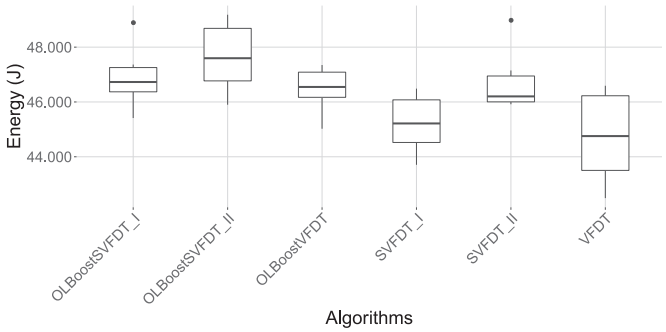


Fig. 4. Boxplots of energy usage of VFDT, SVFDT-I and SVFDT-II, and their versions with OLBoost.

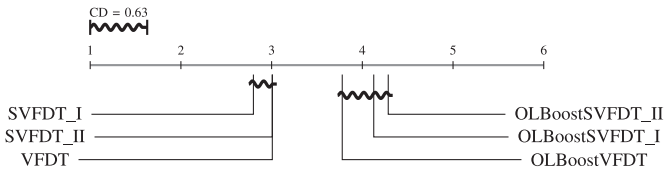


Fig. 5. Comparison of energy usage among VFDT, SVFDT-I and SVFDT-II, and their versions with OLBoost according to Nemenyi test with $\alpha = 0.05$.

four boxplots (one for each metric) using the mean performance metric obtained by each algorithm across all datasets while varying the GP and τ values. Additionally, Friedman's statistical test and the post-hoc Nemenyi test were employed to verify the statistical differences. CD diagrams were created with these results and ranked in order to show the better algorithms for that metric in the first positions of the ranking and the worse algorithms in the last positions.

B. Time Costs

Figure 2 shows the boxplot for time results per algorithm.

According to the reported results, VFDT was less time consuming than SVFDTs. This is mostly related to the fact that SVFDT enforces additional computations each time a leaf tries to turn into a split node. It is interesting to note that SVFDT-II presented very high time variations. This indicates that SVFDT-II may be more suggestible to the used hyperparameters, varying from being as fast as VFDT to as slower as SVFDT-I. When adding OLBoost to the ODTs algorithms, time was greatly increased. It demonstrates that time is directly impacted by the additional updates employed by the OLBoost strategy. The high time variations of SVFDT-II were also found for its version with OLBoost.

The time analysis of the ODT algorithms was reinforced by the statistical tests, as exposed in Figure 3, which presents the CD diagram for time.

Considering Figure 3, VFDT has a statistical difference to the other algorithms, being the fastest. There were no significant differences between SVFDTs, the second best in the ranking. The OLBoost versions were connected among them by the CD and placed as last in the ranking, showing that OLBoost statistically implied in more time usage.

C. Energy Consumption

Figure 4 shows the boxplot for the energy metric. The results are shown for the six algorithms.

VFDT had the lowest median value for energy consumption. SVFDT-I presented a similar performance, but with a narrower box. Among the algorithms without OLBoost, SVFDT-II is the one that required more energy. When using OLBoost, the energy consumption of all algorithms rose: the first quartiles of VFDT and SVFDT-I with OLBoost were higher than VFDT and SVFDT-I's third quartile. SVFDT-II with OLBoost was the most costly combination in terms of energy among the 6 algorithms in absolute values.

The comparison between Figures 2 and 4 suggests that SVFDT-I operations without OLBoost used less total power than SVFDT-II (also without OLBoost) operations. As Equation 3 shows, if P increases (or is constant) and t increases, E also increases. In Figure 2, the median time for SVFDT-I increased in relation to SVFDT-II. In contrast, in Figure 4, the median energy of SVFDT-I decreased in relation to SVFDT-II. Thus, as time increased and energy decreased, the total power mandatorily decreased. An analogous analysis can be made for the algorithms versions with OLBoost.

The statistical tests illustrated in Figure 5 confirm OLBoost increased the energy consumption. The algorithms without OLBoost consumed less energy than with OLBoost and presented no significant difference among them.

D. Accuracy

Figure 6 shows the boxplot for accuracy results of the six algorithms.

When considering accuracy, VFDT, SVFDT-I, and SVFDT-II had similar performance, with a median of around 0.78. When using OLBoost, the accuracy of the three algorithms increased, resulting in around 0.79.

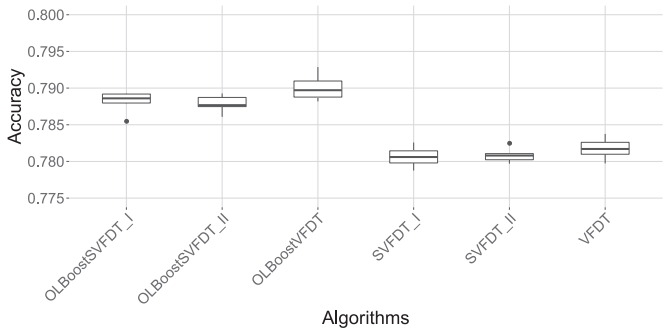


Fig. 6. Boxplots of accuracy of VFDT, SVFDT-I and SVFDT-II, and their versions with OLBoost.

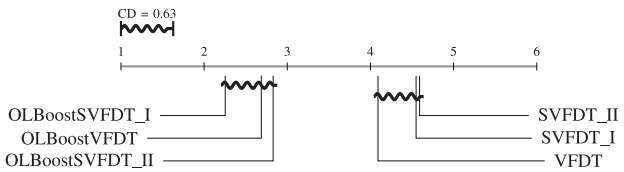


Fig. 7. Comparison of accuracy performance among VFDT, SVFDT-I and SVFDT-II, and their versions with OLBoost according to Nemenyi test with $\alpha = 0.05$.

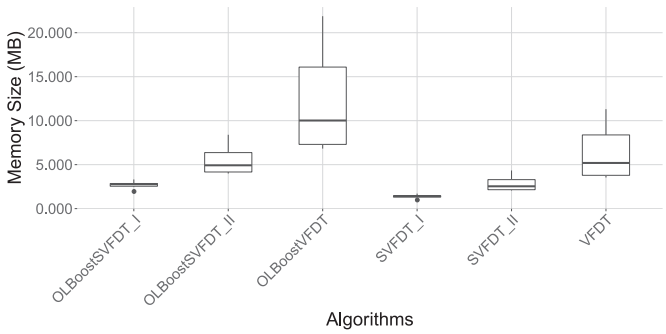


Fig. 8. Boxplots of memory usage of VFDT, SVFDT-I and SVFDT-II, and their versions with OLBoost.

Although this difference was around only 0.01, Figure 7 shows that OLBoost statistically increased the accuracy performance, since the first group connects the three algorithms with OLBoost and the second group contains the three algorithms without OLBoost.

E. Memory Consumption

Figure 8 shows the boxplot of memory results of the six algorithms.

SVFDT-I was the most memory conservative, with a median memory usage of less than 2.5 MB. SVFDT-II was slightly worse, followed by VFDT. It is important to note that both SVFDTs, for all hyperparameter configurations, consumed much less memory than any VFDT. Although this difference (in amount of MBs) may not seem significant at first sight, it must be considered that these devices have low computational resources, so any amount of memory saved is beneficial. OLBoost brought little overhead to these algorithms, especially for VFDT. However, SVFDTs with OLBoost used less memory than VFDT without OLBoost. SVFDTs were already

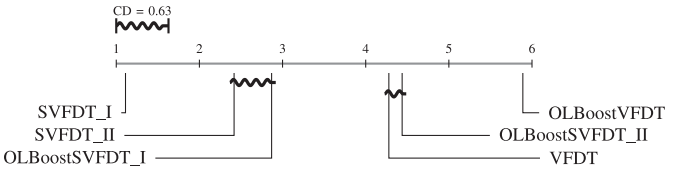


Fig. 9. Comparison of memory performance among VFDT, SVFDT-I and SVFDT-II, and their versions with OLBoost according to Nemenyi test with $\alpha = 0.05$.

expected to use less memory due to the growth control mechanism they are based on. OLBoost works in parallel with the ODT algorithm without using or modifying any statistic used by it. In this sense, OLBoost uses additional memory resources because it needs to store its own statistics to perform predictions and updates when needed. However, this is directly linked to the size of the tree, since the number of extra classifiers used by OLBoost is equal to the number of leaves in the tree. So, the impact of adding OLBoost to smaller decision trees is much lower than to add it to larger trees.

Figure 9 shows the memory consumption CD diagram. Indeed, SVFDT-I was the algorithm that required the least memory. Following it in the ranking, SVFDT-II and SVFDT-I with OLBoost had no statistical difference at a significance level of 5%. After that, SVFDT-II with OLBoost and VFDT did not present a significant difference. At last, VFDT with OLBoost version required the most memory, being the worst case for this metric.

F. Handling the Trade-Off

Considering the EC challenges, the trade-off among the four performance metrics is very important. For example, an algorithm that has a very high predictive performance might not be suitable due to its memory costs.

So far, we explored each performance metric individually. To analyze them concomitantly, we first averaged the results of all datasets for each combination of algorithm, GP and τ. We also took the average between the executions. As an outcome, there were 36 cases that represented the four performance metrics according to GP, τ and algorithm. After that, the data related to energy, time and memory were normalized by dividing all the cases by the case that assumed the highest value in that metric. Thus, the maximum value for a certain metric became 1. The accuracy data were transformed into error by taking $1-$*accuracy*.

Then, we selected the combinations that resulted in the 2 lowest values for each performance metric (best 5%) and created a radar plot to ease the comparison among the metric under analysis and the three remaining ones (for instance, when the best values for energy were selected, the three axes of the radar corresponded to time, error and memory). This was made to evaluate the interaction among the metrics and how prioritizing one component influences the others. Figure 10 presents these radar plots.

For time-costs, VFDT exposed the lowest time consumption with $\tau = 0.10$ and varying the GP between 750 and 1000. The corresponding radar plot shows that similar values can be observed in predictive performance, but energy and memory

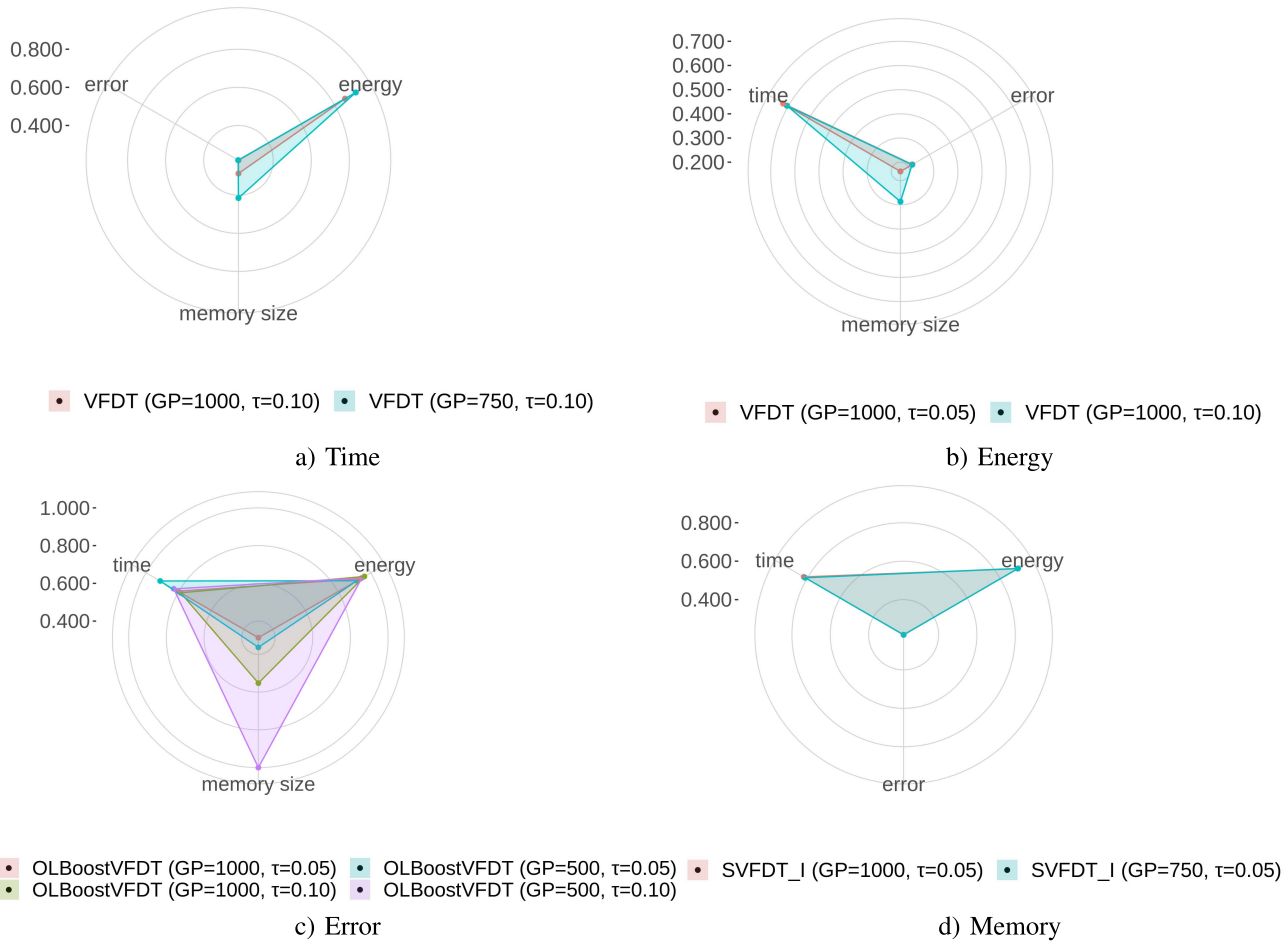


Fig. 10. Interaction of complementary metrics for the best combinations of algorithm, GP and τ when prioritizing time (a), energy (b), error (c) and memory (d).

requirements differ for the different values of GP. The polygon areas in this situation were:

- 0.215 for VFDT (GP $= 1000$, $\tau = 0.10$);
- 0.294 for VFDT (GP $= 750$, $\tau = 0.10$).

Thus, from low time cost perspective, the usage of VFDT with GP=1000 and $\tau = 0.10$ is suggested.

For energy consumption, two combinations delivered the lowest results: VFDT with $GP = 1000$ and $\tau = 0.05$ and VFDT with $GP = 1000$ and $\tau = 0.10$. Analyzing the complementary metrics, for time and error, both cases had similar performance. On the other hand, the memory size when using $\tau = 0.05$ was smaller. For each polygon, the areas corresponded to:

- 0.181 for VFDT (GP $= 1000$, $\tau = 0.10$);
- 0.135 for VFDT (GP $= 1000$, $\tau = 0.05$).

In this sense, when prioritizing lowest energy consumption, VFDT with GP $= 1000$ and $\tau = 0.05$ would be the most recommended.

Taking into account error, it was not possible to select only 2 configurations because 4 configurations reached the same lowest error: VFDT coupled to OLBoost with GP $= 1000$ and $\tau = 0.05$ and $\tau = 0.10$, and VFDT coupled to OLBoost with GP $= 500$ and $\tau = 0.05$ and $\tau = 0.10$. Looking at the corresponding radar plot, it is observable that memory size represented the biggest difference among them, influencing in the areas of the polygons, which were:

- 0.746 for VFDT with OLBoost (GP $= 1000$, $\tau = 0.10$);
- 0.561 for VFDT with OLBoost (GP$= 1000$, $\tau = 0.05$);
- 0.649 for VFDT with OLBoost (GP $= 500$, $\tau = 0.05$);
- 1.109 for VFDT with OLBoost (GP $= 500$, $\tau = 0.10$).

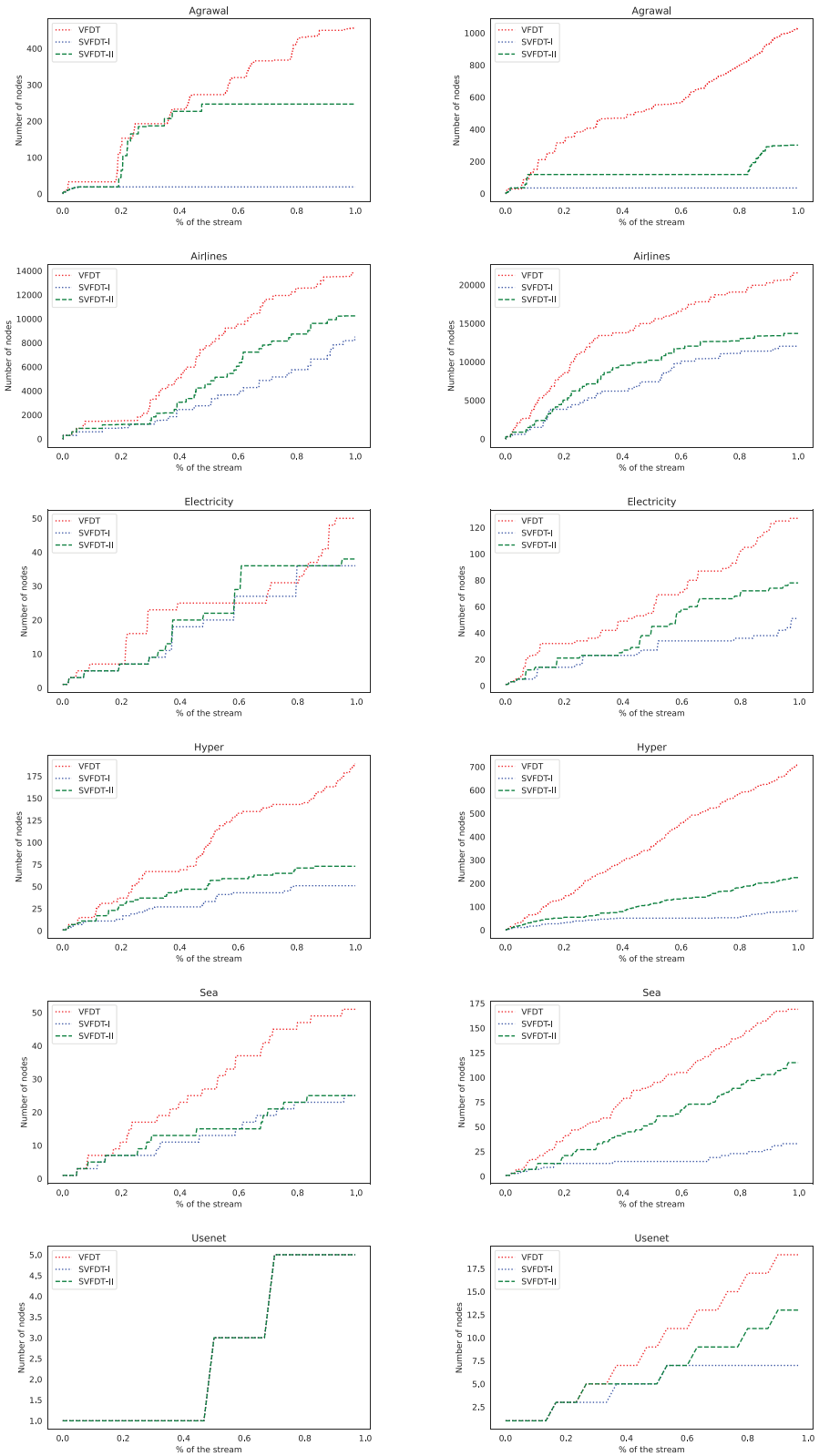
Based on the areas, it is also possible to infer that the prioritization of lower error (i.e., higher accuracy) leads to the choice of VFDT with OLBooost, using GP $= 1000$ and $\tau = 0.05$ as hyperparameters values.

Next, the combinations that demonstrated the lowest requirements of memory size were SVFDT-I varying the GP between 750 and 1000, and maintaining $\tau = 0.05$. The area values of the polygons presented a slight difference:

- 0.480 for SVFDT-I (GP $= 750$, $\tau = 0.05$);
- 0.484 for SVFDT-I (GP $= 1000$, $\tau = 0.05$).

Regarding lower memory size preference (which is a fundamental specification of hardware selection), the most appropriate choice would be SVFDT-I with GP $= 750$ and $\tau = 0.05$.

Summarizing the presented results, for each performance metric, the best configurations of algorithm and hyperparameters were observed. When the metrics energy, time, or predictive performance were prioritized, the complementary metrics were mostly similar among them for the different best



a) GP=1000 and τ=0.05 b) GP=500 and τ=0.1

Fig. 11. Tree growth according to the percentage of the stream for GP $= 1000$ and $\tau = 0.05$ (a) and GP $= 500$ and $\tau = 0.1$ (b). Since OLBoost does not influence tree growth, they have the same number of nodes as the version without OLBoost.

configurations. Memory, however, when figuring as a complementary metric, expressed a high variability. In general, VFDT had a distinguished performance for time cost, energy and predictive performance. As for memory cost, as reinforced by Figures 8 and 9, SVFDT-I demonstrated to be the most memory conservative among the algorithms applied in the experiments.

Considering the GP and τ configurations related to Figure 10, it is observable that the combination GP $= 1000$ and $\tau = 0.05$ appears among the best ones for three performance components, and results in the smallest area among all, when prioritizing energy consumption. This configuration combines the highest experimented GP with the lowest τ, which indicates fewer split attempts. Thus, we further analyzed this configuration in terms of tree size. To this end, in Figure 11, we compared the tree growth according to the amount of processed stream for this configuration (GP $= 1000$ and $\tau = 0.05$), and GP $= 500$ and $\tau = 0.1$, which is the combination of the lowest experimented GP and highest τ.

In general, the number of tree nodes generated by SVFDT-I (both with or without OLBoost) was lower than the ones generated by the other algorithms. VFDT, in its turn, is the algorithm that predominantly produced larger trees, independently of the hyperparameter configuration or dataset.

The most noticeable difference in final tree size occurs for Agrawal dataset: whereas the tree generated by SVFDT-I has less than 50 nodes (both in a) and b)), SVFDT-II generated a tree with around 250 nodes (both in a) and b)), and VFDT continued to grow until the end of the stream, reaching more than 450 nodes in a) and 1000 nodes in b). SVFDTs stabilized after some instances of this dataset were processed: for a), SVFDT-I stabilized after around 5% of the stream was processed and SVFDT-II stabilized after 50% of the stream; for b), SVFDT-I also stabilized after around 5% of the stream was processed, but SVFDT-II grew until 90% of the stream was processed, even though it did not grow from 10% to around 80% of the stream.

The Airlines dataset resulted in the largest trees among the 6 datasets. With VFDT, the tree had almost 14000 nodes with configuration a) and more than 20000 nodes with configuration b). SVFDT-I resulted in smaller trees, with 8000 nodes for a) and more than 10000 nodes for b). These examples confirm the impact that hyperparameter and algorithm selection has on the tree size.

In the Electricity dataset, it is observable that in configuration a), when 20% of the stream is processed, VFDT begins to produce a larger tree than the others, but when 60% of the stream is processed, SVFDT-II begins to generate the largest tree. However, when 80% of the stream is processed, VFDT returns to produce the largest tree, ending up with around 50 nodes against around 37 nodes for the SVFDTs. In configuration b), VFDT is the largest tree all the time after the initialization period, with a final tree with more than 120 nodes.

Regarding the Hyper dataset, it reinforces the tree growth control of SVFDTs, especially of SVFDT-I. In fact, the final trees of VFDT are larger than twice the size of SVFDTs for both configurations.

The Sea dataset shows a difference between the SVFDTs. In configuration a), both versions of SVFDT conclude the tree growth with around 25 nodes. In configuration b), the final SVFDTs' trees have different sizes: SVFDT-I has around 25 nodes and SVFDT-II has around 115 nodes.

For the Usenet dataset, all the trees had the same size in configuration a). The fact that this dataset presented the lowest amount of examples and a high GP is a hypothesis for that. In configuration b), the trees remain equal until 20% of the stream is processed. After that, SVFDT-I stabilizes after around 50% of the stream is processed, but the other trees continue to grow until the end of the stream.

Summing up, higher GP values coupled with smaller τ tended to produce smaller trees, and VFDT tended to produce larger trees than SVFDTs. This ratifies the theory and stresses the hyperparameter selection importance, as well as algorithm choice.

VI. Conclusion and Future Work

In this work, we investigated energy consumption, predictive performance, memory and time costs of three different algorithms for data stream classification and the influence of a promising boosting strategy called OLBoost. Energy and memory consumption are specifically important for green computing and some EC application domains, which rely on devices with constrained computational resources and batteries. Additionally, considering that fast algorithms with high predictive performance are also desired for EC, these four metrics should be considered and their choice should be balanced. The experiments were carried out comparing VFDT, SVFDT-I and SVFDT-II with the usage of OLBoost in six datasets with different hyperparameter settings in a Raspberry Pi.

Our results corroborate that the best algorithm depends on the performance metric prioritization: if energy, VFDT is the most advantageous; if time, the version of the algorithms without OLBoost; if accuracy, the use of OLBoost is recommended; if memory, SVFDTs are more propitious, specially SVFDT-I. In addition, inducing the online trees with OLBoost demands for resources increase (energy, time and memory) at the cost of reducing error, in general. Another point to consider when designing ODTs for EC applications, alongside classification algorithm and performance metrics, is the careful selection of values for the hyperparameters, since they greatly impact the tree size.

In this work, we only tested Raspberry Pi as an EC device. Future works could compare the performance in a scenario composed of different devices. Another possible continuing work is the evaluation of other families of data stream classification algorithms. Moreover, although they cannot be directly compared with the results of this work, non-supervised approaches could also be evaluated in this type of hardware.

References

[1] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan./Feb. 2018.

[2] G. Ananthanarayanan *et al.*, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[3] S. Babu, P. V. Mithun, and B. S. Manoj, "A novel framework for resource discovery and self-configuration in software defined wireless mesh networks," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 132–146, Mar. 2019.

[4] X. Yu, C. Cecati, T. Dillon, and M. G. Simões, "The new frontier of smart grids," *IEEE Ind. Electron. Mag.*, vol. 5, no. 3, pp. 49–63, Sep. 2011.

[5] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[6] I. Sittón-Candanedo, R. S. Alonso, J. M. Corchado, S. Rodríguez-González, and R. Casado-Vara, "A review of edge computing reference architectures and a new global edge proposal," *Future Gener. Comput. Syst.*, vol. 99, pp. 278–294, Oct. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X1930264X

[7] B. Krawczyk, L. Minku, J. Gama, and J. Stefanowski, "Ensemble learning for data stream analysis: A survey," *Inf. Fusion*, vol. 37, pp. 1–86, Sep. 2017.

[8] H. M. Gomes *et al.*, "Adaptive random forests for evolving data stream classification," *Mach. Learn.*, vol. 106, no. 9, pp. 1469–1495, Oct. 2017.

[9] J. Gama, *Knowledge Discovery From Data Streams*, 1st ed. Boston, MA, USA: CRC Press, 2010.

[10] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proc. KDD*, 2000, pp. 71–80.

[11] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Proc. 8th Int. Symp. Intell. Data Anal.*, 2009, pp. 249–260.

[12] B. Pfahringer, G. Holmes, and R. Kirkby, "New options for hoeffding trees," in *Proc. 20th Aust. Joint Conf. Adv. Artif. Intell.*, 2007, pp. 90–99.

[13] V. G. Turrisi da Costa, A. C. P. D. L. F. D. Carvalho, and S. Barbon, "Strict very fast decision tree: A memory conservative algorithm for data stream mining," *Pattern Recognit. Lett.*, vol. 116, pp. 22–28, Dec. 2018.

[14] V. G. T. da Costa, S. M. Mastelini, A. C. P. de Leon Ferreira, and S. Barbon, "Online local boosting: Improving performance in online decision trees," in *Proc. IEEE 8th Brazil. Conf. Intell. Syst. (BRACIS)*, 2019, pp. 132–137.

[15] V. G. T. da Costa, S. M. Mastelini, A. C. D. L. de Carvalho, and S. Barbon, "Making data stream classification tree-based ensembles lighter," in *Proc. IEEE 7th Brazil. Conf. Intell. Syst. (BRACIS)*, 2018, pp. 480–485.

[16] V. G. T. da Costa, E. J. Santana, J. F. Lopes, and S. Barbon, "Evaluating the four-way performance trade-off for stream classification," in *Proc. Int. Conf. Green Pervasive Cloud Comput.*, 2019, pp. 3–17.

[17] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers," *Comput. Netw.*, vol. 130, pp. 94–120, Jan. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128617303778

[18] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, "Fog computing May help to save energy in cloud computing," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1728–1739, May 2016.

[19] M. Selimi, A. Lertsinsrubtavee, A. Sathiaseelan, L. Cerdà-Alabern, and L. Navarro, "PiCasso: Enabling information-centric multi-tenancy at the edge of community mesh networks," *Comput. Netw.*, vol. 164, Dec. 2019, Art. no. 106897. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128618312787

[20] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Mach. Learn.*, vol. 85, no. 3, p. 333, 2011.

[21] G. De Francisci Morales, A. Bifet, L. Khan, J. Gama, and W. Fan, "IoT big data stream mining," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2016, pp. 2119–2120.

[22] S. Albers, "Energy-efficient algorithms," *Commun. ACM*, vol. 53, no. 5, pp. 86–96, 2010.

[23] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park, "Advanced lightweight encryption algorithms for IoT devices: Survey, challenges and solutions," *J. Ambient Intell. Human. Comput.*, vol. 8, pp. 1–18, May 2017.

[24] W. Vereecken, W. Van Heddeghem, D. Colle, M. Pickavet, and P. Demeester, "Overall ICT footprint and green communication technologies," in *Proc. IEEE 4th Int. Symp. Commun. Control Signal Process. (ISCCSP)*, 2010, pp. 1–6.

[25] E. Garcia-Martin, N. Lavesson, and H. Grahn, "Energy efficiency analysis of the very fast decision tree algorithm," in *Trends in Social Network Analysis* (Lecture Notes in Social Networks), R. Missaoui, T. Abdessalem, and M. Latapy, Eds. Cham, Switzerland: Springer, 2017, pp. 229–252.

[26] S. Harizopoulos, M. Shah, J. Meza, and P. Ranganathan, "Energy efficiency: The new holy grail of data management systems research," 2009. [Online]. Available: https://arxiv.org/abs/0909.1784

[27] V. H. Bezerra, V. G. T. da Costa, S. B. Junior, R. S. Miani, and B. B. Zarpelão, "IoTDS: A one-class classification approach to detect botnets in Internet of Things devices," *Sensors*, vol. 19, no. 14, p. 3188, 2019.

[28] R. Chen, J. Guo, D.-C. Wang, J. J. Tsai, H. Al-Hamadi, and I. You, "Trust-based service management for mobile cloud IoT systems," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 1, pp. 246–263, Mar. 2019.

[29] O. Ibidunmoye, A.-R. Rezaie, and E. Elmroth, "Adaptive anomaly detection in performance metric streams," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 217–231, Mar. 2017.

[30] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Mach. Learn.*, vol. 90, no. 3, pp. 317–346, 2013.

[31] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Min. Knowl. Disc.*, vol. 30, no. 4, pp. 964–994, 2016.

[32] T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning from streaming data with concept drift and imbalance: An overview," *Progr. Artif. Intell.*, vol. 1, no. 1, pp. 89–101, 2012.

[33] J. a. Gama, R. Rocha, and P. Medas, "Accurate decision trees for mining high-speed data streams," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min. (KDD)*, 2003, pp. 523–528.

[34] G. Holmes, K. Richard, and B. Pfahringer, "Tie-breaking in hoeffding trees," in *Proc. ECML/PKDD*, 2005, pp. 107–116.

[35] H. Yang and S. Fong, "Incremental optimization mechanism for constructing a decision tree in data stream mining," *Math. Problems Eng.*, vol. 2013, Jan. 2013, Art. no. 580397. [Online]. Available: https://doi.org/10.1155/2013/580397

[36] N. C. Oza, "Online bagging and boosting," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, vol. 3, Oct. 2005, pp. 2340–2345.

[37] R. Agrawal, A. Swami, and T. Imielinski, "Database mining: A performance perspective," *IEEE Trans. Knowl. Data Eng.*, vol. 5, no. 6, pp. 914–925, Dec. 1993.

[38] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive online analysis," *J. Mach. Learn. Res.*, vol. 11, no. 52, pp. 1601–1604, 2010.

[39] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min. (KDD)*, vol. 4, pp. 377–382, 2001.

[40] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Tracking recurring contexts using ensemble classifiers: An application to email filtering," *Knowl. Inf. Syst.*, vol. 22, no. 3, pp. 371–391, 2010.

[41] L. Pietruczuk, L. Rutkowski, M. Jaworski, and P. Duda, "How to adjust an ensemble size in stream data mining?" *Inf. Sci.*, vol. 381, pp. 46–54, Mar. 2017.

[42] *PowerTop*. Accessed: Jan. 1, 2019. [Online]. Available: https://01.org/powertop

[43] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, "A preliminary study of the impact of software engineering on GreenIT," in *Proc. 1st IEEE Int. Workshop Green Sustain. Softw. (GREENS)*, 2012, pp. 21–27.

[44] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *ACM SIGOPS Oper. Syst. Rev.*, vol. 47, no. 3, pp. 42–49, 2013.

[45] E. García Martín, "Energy efficiency in machine learning: A position paper," in *Proc. 30th Annu. Workshop Swedish Artif. Intell. Soc. (SAIS)*, vol. 137, 2017, pp. 68–72.

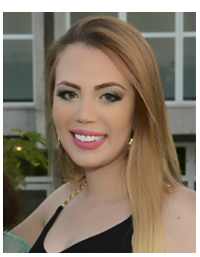
[46] M. G. Patterson, "What is energy efficiency? Concepts, indicators and methodological issues," *Energy Policy*, vol. 24, no. 5, pp. 377–390, 1996.

[47] J. P. Barddal and F. Enembreck, "Learning regularized hoeffding trees from data streams," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput.*, 2019, pp. 574–581.

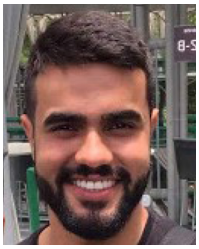
[48] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Disc. Data Min. (KDD)*, 2015, pp. 59–68.

[49] E. Mouw. (2001). *Linux Kernel Proofs Guide*. [Online]. Available: http://lib.hpu.edu.vn/handle/123456789/21423

[50] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.



Jessica Fernandes Lopes received the B.Sc. degree in electrical engineering from the Federal University of Technology–Paraná (UTFPR), Brazil, in 2015, researching in machine learning applied at manufacturing system, and computer engineering, and the Master of Business Administration degree dealing with big data and machine learning to assist in making a decision in 2017. She is currently pursuing the M.Sc. degree in computer science with the State University of Londrina (UEL), Brazil, researching computer vision and pattern Recognition. Her research interests include computer vision, machine learning, and pattern recognition.



Everton Jose Santana received the B.Sc. degree in electrical engineering from the State University of Londrina (UEL), Brazil, in 2019, where he is currently pursuing the M.Sc. degree in computer science. From 2015 to 2016, he was an exchange student with the Hanze University of Applied Sciences, The Netherlands, where he followed minors in biomedical and sensor system engineering. His main research topics are machine learning, instrumentation/biomedical engineering, and applied mathematics.

Victor G. Turrisi da Costa received the M.Sc. and B.Sc. degree in computer science from the State University of Londrina (UEL), Brazil, in 2019 and 2017, respectively. He is currently pursuing the Ph.D. degree in computer vision with the University of Trento, Italy. During his masters, he worked on machine learning algorithms applied to data streams and network security. His current research lies at the intersection of machine learning and computer vision and its applications.

Bruno Bogaz Zarpelão received the B.Sc. degree in computer science from the State University of Londrina (UEL), Brazil, and the Ph.D. degree in electrical engineering from the University of Campinas, Brazil. In 2012, he joined UEL, where he is currently an Assistant Professor with the Computer Science Department. From March 2018 to February 2019, he was a Visiting Postdoctoral Researcher with City, University of London. His research interests include security analytics, intrusion detection, and Internet of Things.

Sylvio Barbon Junior received the B.Sc. degree in computer science in 2005, the M.Sc. degree in computational physics from University of São Paulo in 2007, and the M.Sc. degree in computational engineering and Ph.D. degree from IFSC/USP in 2008 and 2011, respectively. In 2017, he was a Visiting Researcher with the University of Modena and Reggio Emilia, Italy, working on multispectral analysis and machine learning. In 2017, he was a Visiting Researcher with the Universita Degli Studi Di Milano, Italy, focused on data stream and process mining. He is currently a Professor of postgraduate and graduate programs and a Leader of the research group that studies machine learning with the Computer Science Department, State University of Londrina (UEL), Brazil. His research interests include digital signal processing, pattern recognition, and machine learning.