2018 7th Brazilian Conference on Intelligent Systems

Making Data Stream Classification Tree-based Ensembles Lighter

Victor G. Turrisi da Costa
Computer Science Department
Londrina State University
Londrina, Brazil
victorturrisi@uel.br

Saulo M. Mastelini
Computer Science Department
Londrina State University
Londrina, Brazil
mastelini@uel.br

André C. P. de L. F. de Carvalho
Institute of Mathematical and Computer Sciences
University of São Paulo
São Carlos, Brazil
andre@icmc.usp.br

Sylvio Barbon Jr.
Computer Science Department
Londrina State University
Londrina, Brazil
barbon@uel.br

***Abstract*—Recently, several classification algorithms capable of dealing with potentially infinite data streams have been proposed. One of the main challenges of this task is to continuously update predictive models to address concept drifts without compromise their predictive performance. Moreover, the classification algorithm used must be able to efficiently deal with processing time and memory limitations. In the data stream mining literature, ensemble-based classification algorithms are a good alternative to satisfy the previous requirements. These algorithms combine multiple weak learner algorithms, e.g., the Very Fast Decision Tree (VFDT), to create a model with higher predictive performance. However, the memory costs of each weak learner are stacked in an ensemble, compromising the limited space requirements. To manage the trade-off between accuracy, memory space, and processing time, this paper proposes to use the Strict VFDT (SVFDT) algorithm as an alternative weak learner for ensemble solutions which is capable of reducing memory consumption without harming the predictive performance. This paper experimentally compares two traditional and three state-of-the-art ensembles using as weak learners the VFDT and SVFDT across thirteen benchmark datasets. According to the experimental results, the proposed algorithm can obtain a similar predictive performance with a significant economy of memory space.**

***Index Terms*—Machine Learning, Data Streams, Ensembles, Light Weight Algorithm**

I. Introduction

Machine Learning (ML) algorithms have been continuously employed to address many data analysis tasks [1]. Recently, there is a growing demand for ML-based solutions able to deal with very large volumes of data, mainly those coming in streams. This imposes a series of constraints and, different from learning from static datasets, which assumes that all training data is available, learning from data streams assumes that new data can arrive at any time and in any amount. Additionally, the current model can become outdated due to the evolving nature of streams, where concept drifts may occur, changing the data distribution in the problem space over time. Therefore, learning from data streams requires continuous adaptation [1], [2]. Besides, since model updating must be fast and the memory available can be limited, a good learning algorithm should be able to efficiently deal with processing time and memory constraints.

Many learning algorithms have been proposed for data stream modelling [3]–[5]. Among them, the Very Fast Decision Tree (VFDT) algorithm [3] is one of the most successful for data stream classification, being capable of inducing a decision tree in an online fashion by taking advantage of a statistical property, the Hoeffding Bound (HB).

According to Krawczyk et al. [1], data stream researchers are shifting their focus to ensemble-based solutions. The predictive performance obtained by these ensembles depend on the strength of their base learners and the statistical correlation between them [6]. Hence, ensembles can use low predictive performance base learners as long as their correlation is low [6]. Thus, learners with similar predictive performance can be used in an ensemble and perform very similar. However, the use of several base learners increases memory costs and is more time consuming. Although VFDT has a low memory cost, learning from data streams can lead to additional tree growth, increasing memory usage and even compromising the application of ensemble solutions on memory-scarce scenarios, e.g., Internet of Things (IoT) devices. Several ensemble algorithms have been proposed in the data stream literature. Oza [7] adapted the idea of bagging and boosting to an online scenario, creating the OzaBag and OzaBoost algorithms. By further extending and exploring this idea, Bifet et al. [8] proposed an ensemble algorithm called Leveraging Bagging (LevBag), which combines the ADWIN [9] with the bagging strategy in [7]. Gomes et al. [10] extended this idea to create less statistically correlated base learners and make the ensemble more robust to concept drifts, proposing the Adaptive Random Forest (ARF). A different approach is due in Brzezinski et al. [11]. The authors proposed the Online

The authors would like to thank CAPES and CNPq (Brazil) for financial support.

978-1-5386-8023-0/18/$31.00 ©2018 IEEE
DOI 10.1109/BRACIS.2018.00089

Accuracy Updated Ensemble (OAUE), which monitors base learners using a fixed window replacing the worst performing.

This paper investigates the use of Strict Very Fast Decision Tree (SVFDT), a recently proposed modification of VFDT, as an alternative base learner for ensembles [12]. SVFDT adapts VFDT to control tree size without compromising predictive performance. By doing so, SVFDT significantly reduces the memory needed, when compared with ensembles of VFDT. Since the predictive performance of the models induced by the SVFDT are very similar, the predictive performance of the ensembles will be related to how the ensembles guarantee low statistical correlation between the base learners.

The main contributions of this work can be summarised as follows:

1) Investigate the use of the SVFDT as base learner for ensemble solutions (OzaBag, OzaBoost, Leveraging Bagging, Online Accuracy Updated Ensemble and Adaptive Random Forests), reducing memory consumption at the cost of minimal loss of predictive performance;
2) Evaluate and statistically compare different ensemble solutions for data streams using the VFDT and both versions of the SVFDT on multiple benchmark datasets.

The remainder of this work is divided as follows: in Section 2, the main aspects of VFDT are presented. Section 3 presents both versions of the SVFDT. Section 4 details the ensembles-based algorithms used in this study. Section 5 contains the experimental setup. The results and discussion are presented in Section 6. The main conclusions and future work proposals are discussed in Section 7.

II. Very Fast Decision Tree

The Very Fast Decision Tree [3] is a tree-based ML algorithm for data streams based on the HB theorem. Consider a continuous variable v, whose values are bounded by the interval $[v_{min}, v_{max}]$, with a range of values $R = v_{max} - v_{min}$, is independently observed n times and, according to these observations, its mean is $\overline{v}$. Thus, the HB theorem states that this variable has an expected mean $\overline{v_{true}}$ (when $n \rightarrow \infty$) bounded by the interval $[\overline{v}-\epsilon, \overline{v}+\epsilon]$ with statistical probability $1-\delta$, where

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

The VFDT follows the HB theorem to split a given leaf during its training phase. After evaluating the candidate features at a split attempt with a heuristic measure $G(.)$ (e.g., Information Gain (IG) or Gini Index (GI)), VFDT uses the HB theorem to check whether the best split candidate would remain the best, had the tree observed more instances. Presuming that the features with the highest and second highest $G(.)$ values are X_{b} and X_{sb}, respectively, let $\Delta G = G(X_{\text{b}}) - G(X_{\text{sb}})$. If $\Delta G > \epsilon$, then X_{b} is the best feature to split, with probability $1-\delta$.

At each leaf, when dealing with a nominal feature, a counting procedure is employed to support the decision. For numeric features, it usually employs a Gaussian estimator, since it is well balanced between yielding the best splits and using low memory [5]. Based on these assumptions, VFDT builds a model from a single instance at a time using limited computational memory resources. Additionally, under realistic assumptions, it has the same asymptotic performance as a decision tree produced by a standard batch algorithm [2].

Split condition analysis occurs whenever a given leaf has more than one class of instances and in intervals of n instances falling into that leaf, reducing unnecessary computation [3]. A tiebreak hyperparameter τ supports tree growth when ΔG is very low. This is performed by checking if $\Delta G < \epsilon < \tau$ is true, ignoring the HB condition [3].

Gama et al., in [13], suggested the use of Naive Bayes (NB) or Adaptive Naive Bayes (ANB) instead of using a traditional most common (MC) prediction at the leaves to increase the VFDT predictive performance.

III. Strict Very Fast Decision Tree

The SVFDT algorithm is a modification of the VFDT proposed by Costa et al. [12] with the goal of creating trees smaller trees than VFDT, but with the same predictive performance. The authors proposed two versions, the SVFDT-I and SVFDT-II, both following the assumptions that:

1) A leaf node should split only if there is a minimum uncertainty of class assumption associated with the instances, according to previous and current statistics;
2) All leaf nodes should observe a similar number of instances to be turned into split nodes;
3) The feature used for splitting should have a minimum relevance according to previous statistics.

Entropy (H) or Information Gain (IG) support the first and third assumptions and the computation of split feature candidates.

To avoid unnecessary growth, SVFDT applies additional rules to hold 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.

To see how it works, consider that each time a leaf satisfy the conditions imposed by the VFDT (according to the HB and tiebreak), the statistics computed at that moment are marked with a *satisfiedVFDT*. For example, the Entropy of that leaf would be marked as $H_{satisfiedVFDT}$ and the $G(.)$ value of the best feature would be $G_{satisfiedVFDT_0}$. The SVFDT splits a leaf when it satisfies all the conditions imposed by the VFDT and four additional constraints applied to each leaf l:

1) $\varphi(H_l, \{H_{l_0}, ..., H_{l_L}\})$,
2) $\varphi(H_l, \{H_{satisfiedVFDT_0}, ..., H_{satisfiedVFDT_S}\})$,
3) $\varphi(G_l, \{G_{satisfiedVFDT_0}, ..., G_{satisfiedVFDT_S}\})$,
4) $n_l \geq \{n_{satisfiedVFDT_0}, ..., n_{satisfiedVFDT_S}\}$,

where L is the total number of leaves at that moment and S is the total number of split attempts that satisfied the VFDT

constraints and n is the number of elements seen at a given leaf.

The SVFDT-II version uses additional ϖ functions as a skipping mechanism to speed-up growing by ignoring all φ functions. The ϖ function is described as:

$$\varpi(x, X) = \begin{cases} \text{True}, & \text{if } x \geq \overline{X} + \sigma(X) \\ \text{False}, & \text{otherwise} \end{cases} \quad (3)$$

At a split attempt, if:

1) $\varpi(H_l, \{H_{satisfiedVFDT_0}, ..., H_{satisfiedVFDT_S}\})$ or
2) $\varpi(G_l, \{G_{satisfiedVFDT_0}, ..., G_{satisfiedVFDT_S}\})$

hold true, then all the other φ constraints are ignored.

The SVFDT memory costs for constraints 2, 3 and 4 are $O(1)$. For constraint 1, it is $O(L_{max})$, with L_{max} being the maximum number of leaves evaluated in the tree induction. Considering time complexity, constraint 1 has a cost of $O(L_{max})$, while the others have $O(1)$ complexity. It is important to highlight that these costs correspond to a single operation and, therefore, the time complexities added to the whole induction process are $O(S * L_{max})$ and $O(S)$, respectively, where S is the number of times a leaf satisfied the VFDT split conditions. For SVFDT-II, there is an additional time cost of $O(S)$ for each mechanism. Despite these additional costs, the tree size is effectively reduced, making SVFDT training similar to, and in some cases faster than VFDT.

IV. Ensembles for Data Streams

Two of the first ensembles proposed for dealing with data streams, OzaBag and OzaBoost, are presented in Oza [7]. OzaBag is an adaptation of the idea of bagging proposed by Breiman [6] to the online scenario. The bagging strategy has been widely used across the majority of ensembles for data stream mining [7], [8], [10]. Bagging works by fitting each base learner of the ensemble to a subset of instances, with size n, sampled with repetition from the n instances available. As a result, the base learners will have low statistical correlation, increasing the ensemble predictive performance. As suggested by Oza [7], the general number of times each instance will be select by the bagging process using a sampling procedure with repetition approximates a Poisson(1) distribution when $n \to \infty$, where n is the number of instances. Thus, given a new instance, online bagging can be performed by sampling a k value from the Poisson(1) distribution for each base learner and updating that learner with the new instance k times.

In a similar way, OzaBoost adapts boosting to an online scenario. In boosting, the idea is to make each subsequent base learner counterbalance the error of the previous learners. The online boosting performed by OzaBoost introduces a variable λ, which starts with the value 1. For the first base learner, k is chosen according to Poisson(λ). After updating this learner, λ increases if the learner still cannot correctly classify this instance, or decreases, otherwise. Each subsequent learner is updated using the newest λ value.

Leveraging Bagging is an ensemble method proposed by Bifet et al. [8], that combines the online bagging strategy proposed in [7] and the ADWIN algorithm [9]. ADWIN keeps a variable length window of values to monitor changes in the average values inside the window [8], [9]. In the LevBag algorithm, each base learner is coupled with ADWIN to monitor its error rate. When a new instance arrives, each base learner is updated with it. After that, their corresponding ADWINs are updated with 0, if their respective base learner can not correctly predict that instance, or 1 otherwise. If the ADWIN reports a change in the error rate of a base learner and the current error increased, the lowest performing base learner is replaced with a new learner. Additionally, they evaluated the performance with different λ values and observed that the best empirical value was $\lambda = 6$.

Online Accuracy Updated Ensemble [11] maintains a weighted set of base learners and predict the class of incoming instances by aggregating the predictions of base learners using weighted voting. For each new instance, each classifier is weighted according to its accuracy and trained incrementally. Additionally, it substitutes the weakest performing base learner after every n instances. The weight of the base learners is computed estimating the prediction error on the last n instances.

Adaptive Random Forests [10] is one of the newest and better performing ensemble for data streams. It can be viewed as an extension of the LevBag algorithm, with some specific changes. First, it proposes a modification to the decision trees used as default base learner. After performing a split, only m features (default value is $\sqrt{f} + 1$, with f being the total number of features) will be considered as possible split features in the resulting leaves. Next, it uses two ADWINs for each base learner instead of one, one more permissive and another more restricted. The first ADWIN is used to start a background tree, while the second is used to evaluate when to swap the background tree with the tree being currently used. In this sense, when a tree's performance decrease, a new background tree is created to learn simultaneous with the next instances. When the performance decreases even further, the tree being used is replaced by its background tree. The bagging mechanism works in the same way as in the LevBag.

The original works present additional details regarding each ensemble.

In addition to employing multiple base learners (in the majority of cases trees), boosting/bagging strategies may result in faster growing trees, since a single instance can be used multiple times, increasing memory costs. Likewise, some strategies, e.g., LevBag and ARF, employ algorithms such as ADWIN, which has its own memory/time costs.

V. Experimental Setup

In order to compare the use of the SVFDTs as base learners instead of the VFDT, the following 13 benchmark datasets were selected:

- **Agrawal dataset (agrawal)** [14].
- **Airline dataset (airline)** [15].
- **Forest Cover Type dataset (covType)** [15] [10].
- **Electricity Pricing dataset (elec)** [15].

- **Hyper Plane dataset (hyper)** [15].
- **Led datasets**: 10% noise and concept drift on 5 features and 20% noise without drifts (**led_10**, **led_20**) [15].
- **Poker-hand dataset (poker)** [15].
- **Random RBF datasets**: 250 k instances with 50 features, 500 k instances with 10 features and 1 kk instances with 10 features (**rbf_250k**, **rbf_500k** and **rbf_1kk**) [10], [16]
- **SEA dataset (sea)** [17].
- **Usenet dataset (usenet)** [18].

Table I summarises the main aspects of each dataset.

TABLE I
SUMMARY OF THE DATASETS USED IN THE EXPERIMENT.

Dataset	# instances	# features			# classes	% maj. class
		numeric	binary	categorical		
agrawal	1,000,000	6	0	3	2	0.672
airlines	539,383	3	0	4	2	0.554
covType	581,012	10	44	0	7	0.487
elec	45,312	6	0	1	2	0.575
hyper	250,000	10	0	0	2	0.500
leds	1,000,000	0	24	0	10	0.100
poker	829,200	5	0	5	10	0.501
rbf_250k	250,000	50	0	0	2	0.504
rbf_500k	500,000	10				0.536
rbf_1kk	1,000,000					
sea	60,000	3	0	0	2	0.626
usenet	5930	0	658	0	2	0.504

For each dataset, the ensembles were tested using with VFDT, SVFDT-I or SVFDT-II as base learner. When comparing each pair, ensemble and base-learner, the accuracy, Kappa M, processing time and average memory size during training (in MB) were measured. Kappa M [19] was proposed to deal with unbalanced datasets and measures how a classifier compares against a majority class predictor. To minimise variance, prequential learning [2] was performed ten times. The settings used for the base learners are presented in Table II.

TABLE II
HYPERPARAMETERS FOR THE BASE LEARNERS USED IN THE EXPERIMENT.

GP	τ	δ	Numeric Estimator	Leaf Predictor
1000	0.05	10^{-5}	Gaussian - 100 bins	ANB

For the ensembles setup, 10 base learners were used, as in [11]. Additional specific settings were:

1) LevBag without colouring, $\lambda = 6$ for Poisson sampling and $\delta = 0.0001$ for the ADWINs;
2) ARF in its fast setting, $\delta = 0.01$ for the ADWINs used to detect warning and $\delta = 0.001$ for the ADWINs used to detect drifts and at each leaf, $m = \sqrt{f} + 1$ features were being evaluated;
3) For the OAUE, a window of 1000 instances to monitor accuracy was used.

VI. RESULTS AND DISCUSSION

Table III presents the average accuracy, memory consumption and processing time for the ensemble-tree pairs and for the single trees. Values in bold represent the top five best values. It is possible to see that the ARF-VFDT pair obtained the best predictive performance, followed by ARF-SVFDT-II, which tied with LevBag-VFDT. Nonetheless, both ensemble solutions have the highest memory and processing time cost. Thus, the trade-off may not be worth. Additionally, OAUE combined with SVFDT-I used less memory than a single VFDT, while increasing predictive performance. Time costs are stacked by around at least 10-fold, since 10 base learners were employed. If a larger number of base learners were employed, e.g., 100, memory and time costs would largely increase.

TABLE III
AVERAGE PERFORMANCE FOR EACH ALGORITHM ACROSS ALL DATASETS.

Ensemble	Base Learner	Aver. Accuracy	Aver. Mem. (MB)	Aver. Time (s)
OzaBag	VFDT	0.788 ± 0.158	25.79 ± 40.31	**1285.66 ± 932.96**
	SVFDT-I	0.777 ± 0.154	10.03 ± 17.83	1306.18 ± 954.07
	SVFDT-II	0.783 ± 0.157	14.91 ± 24.30	**1293.78 ± 946.52**
OzaBoost	VFDT	0.795 ± 0.157	27.42 ± 43.06	1571.49 ± 1190.32
	SVFDT-I	0.786 ± 0.161	14.49 ± 23.69	1593.92 ± 1228.29
	SVFDT-II	0.789 ± 0.158	18.68 ± 28.20	1581.65 ± 1216.46
LevBag	VFDT	**0.801 ± 0.161**	44.75 ± 43.93	2473.58 ± 1802.60
	SVFDT-I	0.794 ± 0.158	8.68 ± 8.48	2650.24 ± 1812.76
	SVFDT-II	**0.798 ± 0.159**	24.30 ± 24.66	2527.20 ± 1800.68
OAUE	VFDT	0.794 ± 0.156	11.02 ± 12.11	1908.19 ± 1518.20
	SVFDT-I	0.785 ± 0.153	**2.64 ± 2.62**	1920.94 ± 1540.46
	SVFDT-II	0.789 ± 0.154	**5.87 ± 6.540**	1907.69 ± 1529.60
ARF	VFDT	**0.804 ± 0.159**	68.21 ± 57.45	3151.70 ± 2348.47
	SVFDT	**0.799 ± 0.155**	13.33 ± 12.85	3369.85 ± 2391.82
	SVFDT-II	**0.801 ± 0.156**	33.80 ± 28.98	3192.40 ± 2400.02
None	VFDT	0.782 ± 0.152	**2.57 ± 4.07**	**180.86 ± 130.81**
	SVFDT-I	0.771 ± 0.150	**0.92 ± 1.65**	**189.13 ± 137.28**
	SVFDT-II	0.776 ± 0.152	**1.56 ± 2.39**	**184.30 ± 133.29**

Table IV shows the average values of accuracy, Kappa M, memory and processing time considering the three base learners individually, not in an ensemble. The values in bold represent the best result for a specific dataset (or those with a very small difference, 0.001 for accuracy, 0.005 MB for memory, 0.002 for Kappa M and 1 second for time). SVFDT-I created smaller trees for all datasets. Although the SVFDT-II produced trees larger than SVFDT-I, in most datasets they were smaller than those produced by VFDT by a large margin. Regarding accuracy, VFDT only obtained values higher than 1% in four datasets. While, considering processing time, VFDT was slightly faster, 3.45% faster than the SVFDT-I and 1.5% faster than the SVFDT-II, on average. It must be observed that SVFDT performs additional procedures to cope with the constraints to reduce tree growth.

To support the comparison between using a VFDT and SVFDT, the SVFDT relative performance (RP) for accuracy, memory and time were computed. The obtained values are presented as box plots (in Figure 1, by ensemble algorithm. The RP was computed as the ratio between a given performance metric of an ensemble using VFDT versus that same ensemble using SVFDT. Improvements obtained by using the SVFDT are represented by values larger than 1, for accuracy and Kappa M, and smaller than 1, for average memory during growth and processing time. It is possible to observe that accuracy and processing time stayed closely the same, whereas memory largely decreased. The high variation in memory reduction is due to particularities of some datasets. SVFDT-I presented median memory consumption reduction between 45% and 70%, whereas SVFDT-II presented a reduction between 30% and 40%. Despite these reductions, there was a low impact on predictive performance. For both SVFDTs there was a maximum decrease of 0.04 in relative accuracy. Finally, SVFDT processing time was greater at most by 5%,

TABLE IV
PERFORMANCE OF A SINGLE BASE LEARNER IN EACH DATASET.

Dataset	Algorithm	Accuracy	Aver. Mem. (MB)	Kappa M	Time (s)
agrawal	VFDT	0.948	2.841	0.841	**203.36 ± 0.39**
	SVFDT	**0.950**	**0.167**	**0.847**	218.73 ± 0.45
	SVFDT-II	**0.949**	2.579	**0.846**	213.88 ± 0.63
airlines	VFDT	**0.665**	15.678	**0.248**	**237.91 ± 2.72**
	SVFDT	0.662	**6.251**	0.242	255.07 ± 7.13
	SVFDT-II	0.661	9.193	0.238	246.9 ± 1.38
covType	VFDT	**0.719**	2.029	**0.452**	**332.06 ± 0.91**
	SVFDT	0.704	**1.473**	0.423	342.34 ± 0.89
	SVFDT-II	0.706	1.571	0.426	338.87 ± 0.50
elec	VFDT	0.786	0.168	0.497	**8.90 ± 0.40**
	SVFDT	**0.794**	**0.136**	**0.514**	**8.93 ± 0.03**
	SVFDT-II	**0.793**	0.153	**0.512**	**8.89 ± 0.05**
hyper	VFDT	**0.892**	0.614	**0.784**	65.27 ± 3.03
	SVFDT	0.890	**0.219**	**0.781**	**61.89 ± 0.83**
	SVFDT-II	0.887	0.321	0.774	**61.74 ± 0.60**
led_10	VFDT	**0.728**	1.000	**0.698**	**360.81 ± 0.95**
	SVFDT	0.723	**0.492**	0.692	373.44 ± 1.18
	SVFDT-II	**0.727**	0.738	**0.697**	369.33 ± 0.88
led_20	VFDT	**0.500**	0.949	**0.444**	347.04 ± 4.51
	SVFDT	0.491	**0.523**	0.434	366.32 ± 3.91
	SVFDT-II	0.495	0.643	0.438	**339.69 ± 4.45**
poker	VFDT	**0.746**	1.196	**0.492**	**263.02 ± 0.52**
	SVFDT	0.714	**0.583**	0.428	282.38 ± 0.67
	SVFDT-II	0.721	0.945	0.442	284.64 ± 0.57
rbf_250k	VFDT	**0.985**	3.083	**0.970**	**133.87 ± 5.05**
	SVFDT	0.974	**0.468**	0.947	147.27 ± 2.75
	SVFDT-II	**0.984**	1.387	0.967	142.11 ± 2.74
rbf_500k	VFDT	**0.901**	1.557	**0.786**	119.31 ± 0.58
	SVFDT	0.863	**0.237**	0.703	118.42 ± 0.20
	SVFDT-II	0.884	0.610	0.750	**116.23 ± 0.32**
rbf_1kk	VFDT	**0.913**	3.066	**0.811**	256.52 ± 7.48
	SVFDT	0.874	**0.294**	0.728	260.29 ± 1.14
	SVFDT-II	0.895	0.949	0.774	**250.03 ± 0.75**
sea	VFDT	**0.849**	0.093	**0.595**	**10.26 ± 0.04**
	SVFDT	**0.848**	**0.047**	**0.593**	**10.88 ± 0.17**
	SVFDT-II	**0.848**	**0.051**	**0.593**	**10.72 ± 0.16**
usenet	VFDT	**0.537**	**1.119**	**0.067**	**12.88 ± 0.17**
	SVFDT	**0.537**	**1.121**	**0.067**	**12.71 ± 0.20**
	SVFDT-II	**0.537**	**1.121**	**0.067**	**12.88 ± 0.35**

which is expected, since it performs additional computations to reduce memory costs.

To analyse the statistical significance of the results, the Friedman statistical test (α = 0.05) and the posthoc Nemenyi (Critical Distance (CD) = 6.82) analysis were employed to compare predictive performance (Figure 2), memory (Figure 3) and time (Figure 4). As reported in [20], this test is used to compare multiple ML algorithms evaluated across multiple datasets.

Figure 2 shows that there is no significant influence of base learners in the accuracy enhancement, since the better ranked ensemble algorithms varied their base learner. The best algorithms were ARF and LevBag, while OAUE, LevBag, OzaBag and OzaBoost presented similar performances. Although single base learners presented the lowest accuracy, they were statistically comparable to some ensembles, OzaBag, OzaBoost, OAUE and LevBag. Based on these results, it is possible to say that ARF coupled with VFDT or SVFDT-II performed better than a single VFDT with statistical significance.

Concerning memory use (Figure 3), the SVFDT versions were the best ranked. According to the Nemenyi test, the lighter algorithms were always composed by SVFDTs. When VFDT was used as the base learner, the ensembles tend to consume statistically more memory. Considering the CD of 6.82, when OAUE, OzaBag and LevBag employ SVFDTs, their memory consumption were statistically comparable to a single VFDT. Therefore, the usage of the SVFDT can create

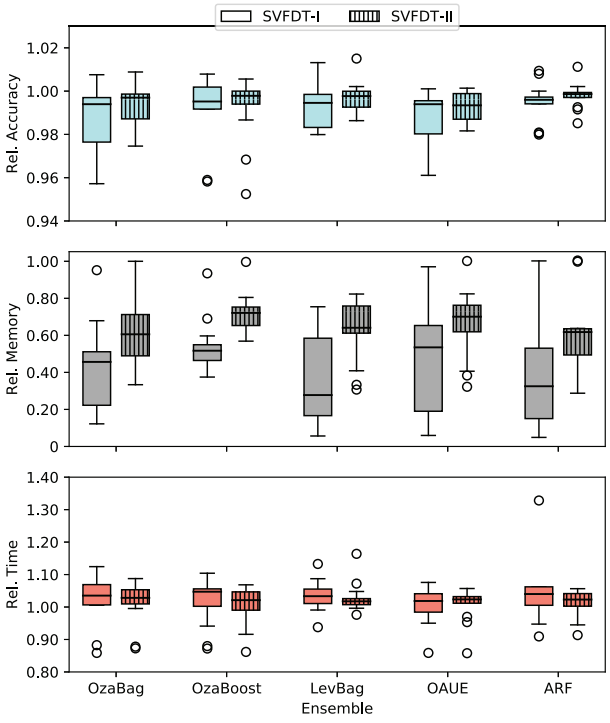

Fig. 1. Boxplots of VFDT relative performance over SVFDT-I and SVFDT-II by ensemble algorithm

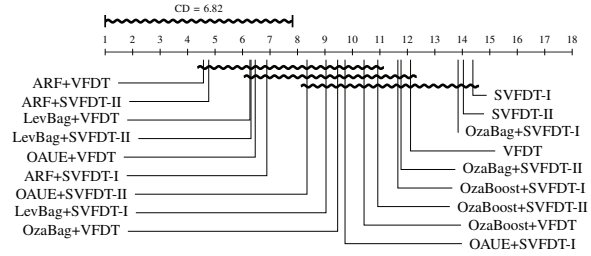

Fig. 2. Critical Distance evaluation of average accuracy with Nemenyi test ($\alpha = 0.05$) considering ensemble techniques and single base learners

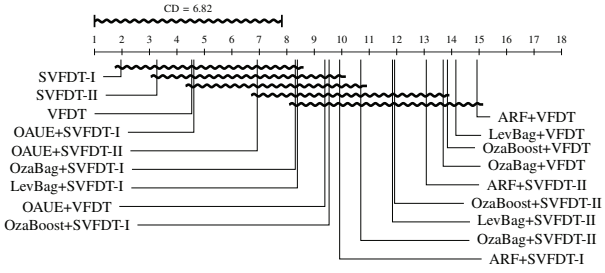

Fig. 3. Critical Distance evaluation of memory consumption with Nemenyi test ($\alpha = 0.05$) considering ensemble techniques and single base learners

an ensemble memory-friendly model as light as that of a single VFDT.

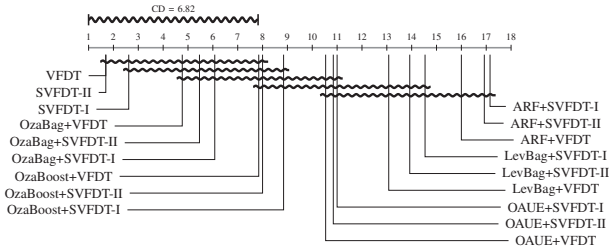

Fig. 4. Critical Distance evaluation of processing time with Nemenyi test ($\alpha = 0.05$) considering ensemble techniques and single base learners

Regarding processing time, single base learner were faster than ensembles, as expected. Likewise, the fastest ensemble algorithms are the simplest (OzaBag and OzaBoost). Indeed, OzaBag was the fastest ensemble algorithm, statistically similar to a single base learner and different from the ARF and LevBag ensembles.

Finally, it was possible to largely reduce average memory costs across all ensemble solutions by using SVFDT. Memory consumption for OzaBag, OzaBoost, LevBag, OAUE and ARF were reduced, on average, by, respectively, around 61%, 47%, 81%, 86% and 80% for SVFDT-I and 42%, 32%, 46%, 47% and 50% for SVFDT-II. These results were obtained reducing predictive performance, on average, by no more 1.1% for SVFDT-I and 0.6% for SVFDT-II, with a small increase in processing time.

VII. Conclusion and Future Work

This work proposed and investigated the use of the SVFDTs as the base learner in ensemble solutions for data stream classification. For such, experiments were carried out comparing, for a set of performance metrics, five ensemble solutions composed by VFDT, SVFDT-I or SVFDT-II as base learners using thirteen datasets. The experiments also compared the performance metrics for individual decision trees. According to the experimental results, the use of both versions of SVFDT resulted in large reductions in memory costs regarding all other ensembles, with a minimal decrease in predictive performance. Although ensembles using the VFDT memory costs were at most 68 MB, the SVFDT reduced reduced this average cost to 13 MB. If more learners, for example 1000, were used, this cost would raise up to 6.6 GB and the SVFDT would reduce it to 1.3 GB. In some cases, the predictive performance even increased. These memory improvements were also present when considering single trees. After applying the Friedman statistical test and posthoc Nemenyi analysis to the experimental results, it was observed that the use of SVFDTs as base learner in ensembles reduced memory need while maintaining predictive accuracy and processing time. As future work, the authors intend to explore more hyper parameters for ensembles and base learners as well as other ensemble solutions coupled with different base learners.

References

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

[2] J. Gama, P. P. Rodrigues, E. Spinosa, and A. Carvalho, "Knowledge Discovery from Data Streams," *Web Intelligence and Security - Advances in Data and Text Mining Techniques for Detecting and Preventing Terrorist Activities on the Web*, pp. 125–138, 2010.

[3] P. Domingos and G. Hulten, "Mining high-speed data streams," pp. 71–80, 2000.

[4] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, pp. 97–106, 2001.

[5] B. Pfahringer, G. Holmes, and R. Kirkby, "Handling numeric attributes in hoeffding trees," in *Proc. of the XII Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD'08, pp. 296–307, 2008.

[6] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct 2001.

[7] N. C. Oza, "Online bagging and boosting," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2340–2345 Vol. 3, Oct 2005.

[8] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part I*, ECML PKDD'10, (Berlin, Heidelberg), pp. 135–150, Springer-Verlag, 2010.

[9] A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing," *Proceedings of the 2007 SIAM International Conference on Data Mining*, pp. 443–448, 2007.

[10] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, vol. 106, pp. 1469–1495, Oct 2017.

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

[12] V. G. T. Costa, A. C. P. L. Carvalho, and S. Barbon Junior, "Strict Very Fast Decision Tree: a memory conservative algorithm for data stream mining," *ArXiv e-prints*, May 2018.

[13] J. a. Gama, R. Rocha, and P. Medas, "Accurate decision trees for mining high-speed data streams," in *Proc. of the IX ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, (New York, NY, USA), pp. 523–528, ACM, 2003.

[14] R. Agrawal, A. Swami, and T. Imielinski, "Database Mining: A Performance Perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.

[15] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: massive online analysis," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, pp. 10–18, Nov. 2009.

[17] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, vol. 4, pp. 377–382, 2001.

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

[19] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *Proc. of the XXI ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, (New York, NY, USA), pp. 59–68, ACM, 2015.

[20] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.