ELSEVIER

Strict Very Fast Decision Tree: a memory conservative algorithm for data stream mining

Victor Guilherme Turrisi da Costa*[a], André Carlos Ponce de Leon Ferreira de Carvalho[b], Sylvio Barbon Junior[a],

[a]*Computer Science Department, State University of Londrina, Londrina - PR, 86057-970, Brazil*
[b]*Institute of Mathematical and Computer Sciences, University of São Paulo - USP, São Carlos - SP, 13566-590, Brazil*

ABSTRACT

Dealing with memory and time constraints are current challenges when learning from data streams with a massive amount of data. Many algorithms have been proposed to handle these difficulties, among them, the Very Fast Decision Tree (VFDT) algorithm. Although the VFDT has been widely used in data stream mining, in the last years, several authors have suggested modifications to increase its performance, putting aside memory concerns by proposing memory-costly solutions. Besides, most data stream mining solutions have been centred around ensembles, which combine the memory costs of their weak learners, usually VFDTs. To reduce the memory cost, keeping the predictive performance, this study proposes the Strict VFDT (SVFDT), a novel algorithm based on the VFDT. The SVFDT algorithm minimises unnecessary tree growth, substantially reducing memory usage and keeping competitive predictive performance. Moreover, since it creates much more shallow trees than VFDT, SVFDT can achieve a shorter processing time. Experiments were carried out comparing the SVFDT with the VFDT in 11 benchmark data stream datasets. This comparison assessed the trade-off between accuracy, memory, and processing time. Statistical analysis showed that the proposed algorithm obtained similar predictive performance and significantly reduced processing time and memory use. Thus, SVFDT is a suitable option for data stream mining with memory and time limitations, recommended as a weak learner in ensemble-based solutions.

1. Introduction

Traditional machine learning (ML) algorithms work by modelling knowledge from static and previously collected datasets. Currently, there is a growing demand for ML-based solutions able to deal with very large volumes of data, which usually comes in the form of continuous streams, creating new challenges. Differently to learning from static data, which assumes that all training data necessary to induce a model is available, learning from data streams assumes that new data can arrive at any time, which can make a model outdated. This may happen due to the occurrence of concept drifts, which are related to the change of data distribution in the problem space over time. Therefore, learning from data streams requires continuous model updates. An additional challenge posed by learning

from data streams is the demand to perform accurate predictions at any time (Krawczyk et al., 2017; Gama et al., 2010). Also, since the model updating must be fast and the memory available can be limited, depending on where it occurs, it is expected that a good algorithm is capable of efficiently dealing with processing time and memory space.

Many learning algorithms have been proposed to cope with some of these aspects. Among them, the Very Fast Decision Tree (VFDT) algorithm (Domingos and Hulten, 2000) is one of the most well-known for stream classification, being capable of constructing a decision tree in an online fashion by taking advantage of a statistical property called Hoeffding Bound (HB). By doing so, the VFDT obtains a predictive performance similar to conventional decision tree induction algorithms applied to static datasets . Although VFDT is somewhat memory-friendly, learning from data streams can lead to unnecessary tree growth, increasing memory usage and even compromising its application on memory-scarce scenarios.

In the last years, (Holmes et al., 2005; Yang and Fong, 2011b, 2013) proposed a series of modifications to increase the predic-

e-mail: victorturrisi@uel.br (Victor Guilherme Turrisi da Costa*),
andre@icmc.usp.br (André Carlos Ponce de Leon Ferreira de Carvalho),
barbon@uel.br (Sylvio Barbon Junior)

tive performance of the VFDT algorithm. However, this came with a substantial increase in the memory cost. Moreover, according to Krawczyk et al. (2017), data stream researchers are shifting their focus to ensemble-based solutions. The performance of these solutions depend on the strength of their base learners and the statistical correlation between them. Hence, ensembles can use only weak learners as long as their correlation is low (Breiman, 2001). Thus, learners with very similar predictive performance could be used as base learners for an ensemble and have virtually the same performance. However, the use of several base-learners increase memory costs, limiting the use of ensembles.

In order to deal with memory cost restrictions, keeping the predictive performance, we propose a new base learner, called Strict Very Fast Decision Tree (SVFDT). Our algorithm addresses these requirements, while being faster than the VFDT in some cases. Thus, SVFDT can cope with memory-scare scenarios and ensemble-based solutions in the following way:

1. SVFDT uses significantly less memory in comparison to VFDT, reaching similar predictive performance;

2. SVFDT and VFDT were compared with various benchmark datasets through critical result analysis;

3. Two SVFDT versions were proposed, one designed to consume less memory and training time (SVFDT-I) and another (SVFDT-II) with a higher predictive performance.

Experiments were performed on various benchmark datasets, measuring the accuracy, Kappa M, memory, and training time of SVFDT and VFDT, and performing a statistical test to assess significant statistical differences.

This paper is organised as follows. Section 2 describes the VFDT algorithm. Section 3 presents other ML algorithms for data streams similar to the VFDT. Section 4 introduces the SVFDT, along with its pseudocode. Section 5 has an empirical study, comparing the proposed algorithm with VFDT, and discussing the results obtained. Finally, Section 6 covers the conclusion and future work.

2. Very Fast Decision Tree

VFDT (Domingos and Hulten, 2000) is a tree-based ML algorithm for data streams designed around the principles of the HB. The HB theorem states the following. Suppose a continuous variable v, whose values are bounded by the interval $[v_{min}, v_{max}]$, with a range of values $R = v_{max} - v_{min}$. Additionally, presume that this variable was independently observed n times and the computed mean, according to these observations, is $\bar{v}$. Thus, the HB theorem states that this variable has a true mean $\overline{v_{true}}$ (when $n \rightarrow \infty$) bounded by the interval $[\bar{v} - \epsilon, \bar{v} + \epsilon]$ with statistical probability $1 - \delta$, where

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

The VFDT algorithm applies the HB to evaluate if a given leaf should be split during the training phase. After ranking

split candidate features during a split attempt, according to a heuristic measure $G(.)$, VFDT uses the HB to check if the best split candidate would remain the best, had the tree observed more instances. Assuming that the features with the highest and second highest $G(.)$ values are X_b and X_{sb}, respectively, let $\Delta G = G(X_b) - G(X_{sb})$. If $\Delta G > \epsilon$, then X_b holds as the best, with probability $1 - \delta$. The $G(.)$ estimates correlation or dependence between two quantities, using metrics, such as Information Gain (IG) or Gini Index (GI).

Based on these assumptions, the VFDT is able to learn 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 (Gama et al., 2010). It is also worth mentioning that the VFDT, unlike batch decision tree induction algorithms, is capable of predicting new instances at any time.

The first version of the VFDT only handled nominal features. Afterwards, many estimators for continuous features were proposed. Pfahringer et al. (2008) reviewed these estimators and observed that the Gaussian estimator is the least sensitive to hyperparameter value and induced the most accurate models, becoming the default estimator in recent works.

To avoid unnecessarily split condition analysis, this checking is only executed if the leaf has an impure class distribution, i.e., there is more than one class of instances that fell on a given leaf. Likewise, with the same goal, this check is only performed after n instances fell into that leaf since the last check. (Domingos and Hulten, 2000). The authors also introduced a tiebreak hyperparameter τ to support tree growth when ΔG is very low. This is done by checking if $\Delta G < \epsilon < \tau$ is true, ignoring the HB condition (Domingos and Hulten, 2000). It must be observed that a high value of τ may lead to tree size explosion and even completely ignoring the HB condition, e.g., when learning from a stream with two classes, using $n = 200$ and $\delta = 10^{-7}$, if $t \geq 0.201$ than the HB condition will never be checked.

Later, to increase VFDT predictive performance, instead of using a traditional most common (MC) prediction at the leaves, a Naive Bayes (NB) or Adaptive Naive Bayes (ANB) algorithm can be employed (Gama et al., 2003).

3. Related Work

Several works proposed modifications to the VFDT algorithm. The Genuine Tie Detection (Holmes et al., 2005) has a mechanism to automatically choose τ during training. Despite the VFDT simplification by removing one hyperparameter, there was a decrease in the predictive performance for most of the datasets used in the experiments.

In a similar work, Yang and Fong (2011a) proposed Optimised-VFDT (OVFDT), whose goal was also to increase accuracy avoiding tree size explosion, substituting τ by statistics about the HB. OVFDT was compared with three algorithms: VFDT (with multiple τ values); Genuine Tie Detection; and Hoeffding Option Tree (HOT) (Pfahringer et al., 2007). It must be observed that none of the compared algorithms try to reduce tree size. When compared with VFDT with $\tau = 0.05$ (VFDT-0.05), OVFDT obtained a small accuracy improvement (3%) at the cost of creating trees 2.4 times larger.

Yang and Fong (2011b, 2013) extended the OVFDT adding statistical constraints related to leaf accuracy. When compared with VFDT-0.05, despite the small improvement in predictive performance, they always produced larger trees.

Other VFDT modification, the Concept-adapting Very Fast Decision Tree (CVFDT) algorithm (Hulten et al., 2001), keeps secondary trees in memory, constantly assessed to check if they outperform the original tree, allowing adaptation to concept drifts. Also, CVFDT uses a sliding window to discard old instances. In the absence of concept drifts, the additional memory costs to store secondary trees makes CVFDT less efficient than VFDT-0.05, as shown in (Yang and Fong, 2011a). In concept drift scenarios, CVFDT predictive performance is much lower than those of ensemble-based solutions (Krawczyk et al., 2017).

Another algorithm based on VFDT, the Hoeffding option tree (HOT) (Pfahringer et al., 2007), includes option nodes, which makes an instance go down into multiple leaves. An option node is essentially a split node with multiple conditions. Thus, a new instance travels along all children nodes whose conditions are true. HOT performs a prediction by averaging the weight of the predictions of all leaves reached. This algorithm presented predictive performance higher than VFDT, at the cost of significant memory increase.

All of these previous modifications to VFDT provided better predictive performance, at the cost of an increase in memory and processing time. Our proposal aims at reducing these drawbacks while keeping a competitive predictive performance. In this way, we evaluate our algorithm using VFDT as the baseline.

4. Strict Very Fast Decision Tree

This section describes the proposed algorithm, Strict Very Fast Decision Tree (SVFDT). SVFDT modifies VFDT by strongly controlling tree growth without degrading predictive performance. We propose two versions of the SVFDT, the SVFDT-I and SVFDT-II. In both versions, the following assumptions hold:

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.

We strongly suggest Entropy (H) and Information Gain (IG) for the first and third assumptions. Likewise, both metrics are also employed to evaluate split feature candidates. However, different functions that work in an analogous way to IG or GI could also be applied.

To avoid unnecessary growth, the following function is adopted, using as an underlying concept the 3-σ rule:

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

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. We assume that X follows a normal distribution.

Additionally, a leaf can satisfy the VFDT split conditions (according to the HB or tiebreak value) and still remain a leaf if SVFDT considers this split unnecessary. When leaves satisfy the VFDT split condition, statistics corresponding to it are marked with an underscored *satisfyVFDT*.

At each leaf l, the following constraints are employed every time there is a split attempt:

1. $\varphi(H_l, \{H_{l_0}, H_{l_1}, ..., H_{l_L}\})$, where the former parameter is the current entropy of l and the latter is a set of all entropies of all current leaves L in the tree, including l (Statement 1);

2. $\varphi(H_l, \{H_{satisfyVFDT_0}, H_{satisfyVFDT_1}, ..., H_{satisfyVFDT_S}\})$, where the latter parameter corresponds to the entropies computed at all S times that a leaf satisfied the VFDT split conditions (Statement 1);

3. $\varphi(IG_l, \{IG_{satisfyVFDT_0}, IG_{satisfyVFDT_1}, ..., IG_{satisfyVFDT_S}\})$, where IG_l is the IG of the best split feature at l and the latter parameter is a set of the IGs computed all S times that a leaf satisfied the VFDT split conditions (Statement 3);

4. $n_l \geq \overline{\{n_{satisfyVFDT_0}, n_{satisfyVFDT_1}, ..., n_{satisfyVFDT_S}\}}$, where the former parameter corresponds to the number of elements seen at l and the latter to average number of elements observed at all S times that a leaf satisfied the VFDT split conditions (Statement 2).

We did not apply the function φ in the last constraint, since it is always possible to satisfy it by waiting for more instances to be assigned to a given leaf. On the contrary, the other constraints are not so easily satisfied in the same way, which may cause deadlocks that even learning a large amount of instances would not resolve.

Additionally to the φ function, SVFDT-II has a skipping mechanism to speed-up growing by ignoring all previously presented constraints using the following function:

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

At a split attempt, if either $\varpi(H_l, \{H_{split_0}, H_{split_1}, ..., H_{split_S}\})$ or $\varpi(IG_l, \{IG_{split_0}, IG_{split_1}, ..., IG_{split_S}\})$ hold true, then all the other φ constraints are ignored.

The memory costs added to VFDT to compute the constraints 2, 3 and 4 are $O(1)$. Complementary, the memory cost of constraint 1 is $O(L_{max})$, with L_{max} being the maximum number of leaves observed during the tree induction.

Regarding time complexity, the first constraint has a cost of $O(L_{max})$, while the others have $O(1)$ complexity. These costs corresponds to a single operation and so, the time complexity added to the whole induction process are $O(t_{satisfiedVFDT} * L_{max})$ and $O(t_{satisfiedVFDT})$, respectively, where $t_{satisfiedVFDT}$ is the number of times a leaf satisfied the VFDT split conditions. For SVFDT-II, we have an additional time cost of $O(t_{satisfiedVFDT})$ for each mechanism. Although we have these additional costs,

Algorithm 1 The SVFDT algorithm.

Input:
    S: the stream of instances
    GP: the grace period
    δ: the error probability
    τ: the tiebreak value
Output:
    $SVFDT$: a trained Strict Very Fast Decision Tree

```
 1: procedure SVFDT(S, GP, δ, τ)
 2:     Let SVFDT ← l_root                              ▷ The root
 3:     Initiate H_statistics, IG_statistics and n_statistics for φ and ϖ equations
 4:     Let LH be the hash of leaves
 5:     Let n_{l_root} ← 0                  ▷ Number of elements seen at l_root
 6:     Let LC_{l_root} ← 0          ▷ Number of elements on last split check at l_root
 7:     Let F_{l_root} ← ∅          ▷ Set of features removed from comparison
 8:     for ((X, y) in S) do    ▷ X is the feature vector of an instance of class y in S
 9:         Sort (X, y) to its leaf l
10:         Let ŷ ← prediction of l
11:         Let n_l ← n_l + 1
12:         Update feature estimators and class distribution at l according to (X, y)
13:         if (class distribution at l is impure ∧ n_l − LC_l > GP) then
14:             Compute HB and IG(.) of features in l ∉ F_l
15:             Let rank ← Sorted IG(.) computed
16:             if (CanSplit(rank, HB, τ, l, LH, H_statistics, IG_statistics, n_statistics)) then
17:                 Remove leaf l from LH
18:                 Replace leaf l with a split node
19:                 for each branch of the split do
20:                     Let l_new ← new leaf
21:                     Initiate all the feature estimators on l_new
22:                     Let class distribution on l_new ← post-split distribution of l_new
23:                     Let n_{l_new} ← sum of class distribution on l_new
24:                     Let LC_{l_new} ← n_{l_new}
25:                     Let F_{l_new} ← ∅
26:                     Add leaf l_new to LH
27:                 end for
28:             else
29:                 Let LC_l ← n_l
30:                 FeatureSelection(rank, HB, F_l)
31:             end if
32:         end if
33:     end for
34:     return SVFDT
35: end procedure
```

Algorithm 2 The split check algorithm.

Input:
    $rank$: sorted list of $IG(.)$ per feature
    HB: the Hoeffding Bound value
    τ: tiebreak value
    l: the current leaf node
    LH: the hash of leaves
    $H_{statistics}$: statistics about entropy values
    $IG_{statistics}$: statistics about IG(.) values
    $n_{statistics}$: statistics about the number of elements seen values
Output:
    Boolean value

```
 1: procedure CanSplit(rank, HB, τ, l, LH, H_statistics, IG_statistics, n_statistics)
 2:     Let IG_best and IG_second_best ← the highest and second highest IG(.)
 3:     if (IG_best − IG_second_best > HB ∨ HB < τ) then
 4:         Compute H̄_LH and σ(H_LH) using LH
 5:         Compute H̄ and σ(H) using H_statistics
 6:         Compute ĪG and σ(IG) using IG_statistics
 7:         Compute n̄ and σ(n) using n_statistics
 8:         Let H_l and n_l ← entropy and number of elements seen at l
 9:         Update H_statistics, IG_statistics and n_statistics with H_l, IG_best and n_l, respectively
10:         Let svfdt_ii_constraints ← H_l ≥ H̄ + σ(H) ∧ IG_best ≥ ĪG + σ(IG)
11:         if (svfdt_ii_constraints) then             ▷ SVFDT-II version only
12:             return True
13:         end if
14:         Let ϱ ← H_l ≥ H̄_LH − σ(H_LH)             ▷ Constraint 1
15:         Let ξ ← H_l ≥ H̄ − σ(H)                    ▷ Constraint 2
16:         Let κ ← IG_best ≥ ĪG − σ(IG)              ▷ Constraint 3
17:         Let ψ ← n_l ≥ n̄ − σ(n)                    ▷ Constraint 4
18:         Let svfdt_constraints ← ϱ ∧ ξ ∧ κ ∧ ψ
19:         if (svfdt_constraints) then
20:             return True
21:         end if
22:     end if
23:     return False
24: end procedure
```

tree size is significantly reduced, making SVFDT training faster or similarly to the VFDT.

Algorithm 1 shows the pseudocode of SVFDT. $H_{\text{statistics}}$, $IG_{\text{statistics}}$, $n_{\text{statistics}}$ and LH correspond to the additional statistics that are used to validate $\varphi(.)$ and $\varpi(.)$ operations. Algorithm 2 implements the function that checks whether a given leaf should be split. In addition to the VFDT split check, we added the variables ϱ, ξ, κ and ψ to denote constraints 1, 2, 3, and 4, respectively. It is worth reminding that all statistics are updated when the VFDT split conditions are satisfied. The procedure of feature selection, invoked in line 30 of Algorithm 1, remained like in (Domingos and Hulten, 2000).

Figure 1 uses a flow chart to illustrate how the VFDT was modified to create the SVFDT-I and SVFDT-II (highlighted in blue).

5. Empirical Study

Both versions of SVFDT were experimentally compared with VFDT using 11 public datasets widely used in the data stream mining literature: 1. Forest Cover Type dataset (cov-Type) (Bifet et al., 2010); 2. Electricity Pricing dataset (elec) (Bifet et al., 2010); 3. Led datasets with 0%, 10% and 20% noise composed of 1 million instances (led_0, led_10, and led_20) (Hall et al., 2009); 4. Random RBF datasets: 10^6 instances with 10 features; 500,000 instances with 10 features; and 250 thousand instances with 50 instances (rbf_1kk,
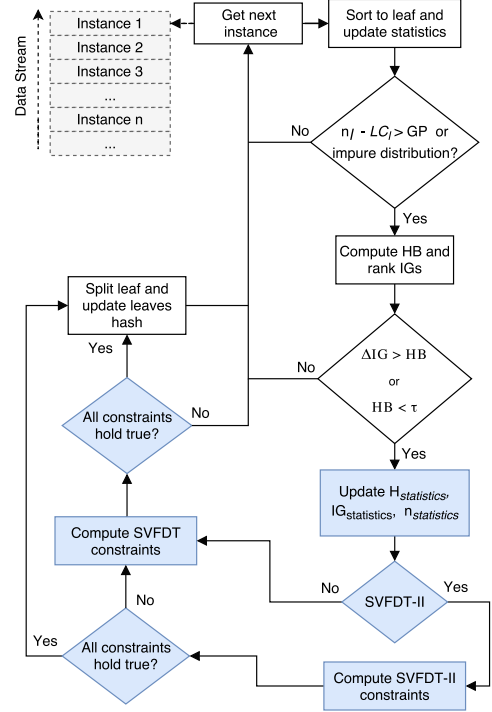

Fig. 1. SVFDT diagram. Parts coloured in blue denote modifications in the traditional VFDT algorithm.

rbf_500k, and rbf_250k(50)) (Hall et al., 2009); 5. SEA dataset (Street and Kim, 2001); 6. Spam dataset (Katakis et al., 2010); 7. Usenet dataset (Katakis et al., 2010). Table 1 briefly describes these datasets.

For each dataset, accuracy and Kappa M (Bifet et al., 2015)

Table 1. Summary of the datasets used in the experiment.

Dataset	# instances	# numeric features	# binary features	# categorical features	# classes
covType	581,012	10	44	0	7
elec	45,312	6	0	1	2
led_0					
led_10	1,000,000	0	24	0	10
led_20					
rbf_1kk	1,000,000	10			
rbf_500k	500,000	10	0	0	2
rbf_250k (50)	250,000	50			
sea	60,000	3	0	0	2
spam	9,324	0	39,917	0	2
usenet	5930	0	658	0	2

measures for the three algorithms were computed, together with the number of tree nodes created. The Kappa M was proposed to deal with unbalanced datasets toward measuring how a classifier compares with another that always predicts the majority class. In the experiments, training time was calculated as the average of 30 runs. Hyperparameter values recommended in the literature were used. These values are shown in Table 2.

Table 2. Hyperparameter values.

GP	τ	Numeric estimator	δ
200	(0.05, 0.10, 0.15, 0.20)	Gaussian with 100 bins	10^{-5}

All algorithms were implemented in Python 3.6, more specifically, VFDT coding was based on MOA's (Bifet et al., 2010).

Table 3 presents the accuracy, Kappa M, tree size and average training time for the four τ values adopted. First, it is possible to observe that the accuracy values for both versions of the SVFDT are very close to those of the VFDT. Likewise, Kappa M values are also close, since they are directly related to accuracy. Regarding the size of the induced trees, it is possible to see a significant discrepancy. In none of the tests performed, the size of SVFDT trees was larger than those of the VFDT trees, with the largest reduction for the rbf_1kk dataset, where the number of nodes decreased from 3194 to 128 (4% of the original size). Excluding the led24_0 dataset, which is very simple and produces very small trees by default, SVFDT largely reduced tree size when compared to VFDT. Finally, one of the main concerns was to avoid impacting training time due to the computation of the new constraints. Although reducing training time is not the focus of this work, in many cases, the smaller trees resulted in shorter training times.

Table 4 shows the relative average accuracy, Kappa M, size and time for each τ value. Relative metrics are obtained by dividing the value obtained by each SVFDT algorithm by the value of VFDT. It is possible to see that, independently from the τ value, accuracy and Kappa M values were very similar. The highest variation occurred when using $\tau = 0.10$, when SVFDT-I and SVFDT-II predictive performances decreased 1.6% regarding VFDT. Considering memory cost, the size of the trees produced by the SVFDT-I were at most 48% of the size of the trees produced by VFDT on average. Although the SVFDT-II produced larger trees than the SVFDT-I, they were at most 67% of the size of the VFDT trees on average. Regarding training time, the SVFDT-I was faster for $\tau = 0.05$ and $\tau = 0.10$, with very significant gains of around 13% to 15%. In contrast with the cases where the VFDT was faster, SVFDT-I was at most 10% slower for $\tau = 0.20$. SVFDT-II was faster than the VFDT only

Table 3. Performance of each algorithm.

Dataset	Algorithm	ACC	Kappa M	Size (nodes)	Time in sec. (std)
covType	VFDT	0.763	0.537	536	167.849 (0.532)
	SVFDT-I	0.758	0.529	365	135.510 (0.662)
	SVFDT-II	0.763	0.537	467	159.249 (1.739)
elec	VFDT	0.801	0.531	209	6.235 (0.025)
	SVFDT-I	0.799	0.526	78	5.594 (0.021)
	SVFDT-II	0.804	0.538	126	6.528 (0.018)
led24_0	VFDT	1.000	1.000	19	27.114 (0.312)
	SVFDT-I	1.000	1.000	19	30.386 (0.436)
	SVFDT-II	1.000	1.000	19	30.217 (0.483)
led24_10	VFDT	0.733	0.703	554	33.918 (0.412)
	SVFDT-I	0.730	0.700	90	34.268 (0.338)
	SVFDT-II	0.731	0.701	255	35.861 (0.228)
led24_20	VFDT	0.504	0.449	524	33.952 (0.463)
	SVFDT-I	0.500	0.444	132	34.228 (0.231)
	SVFDT-II	0.504	0.449	235	35.679 (0.232)
rbf_1kk	VFDT	0.922	0.833	3194	245.774 (0.850)
	SVFDT-I	0.900	0.784	128	169.698 (0.858)
	SVFDT-II	0.909	0.804	864	259.997 (0.987)
rbf_500k	VFDT	0.914	0.815	1746	128.887 (0.361)
	SVFDT-I	0.894	0.771	124	100.163 (0.385)
	SVFDT-II	0.900	0.785	504	138.246 (0.571)
rbf_250k(50)	VFDT	0.990	0.980	455	146.665 (0.331)
	SVFDT-I	0.982	0.964	110	185.454 (0.650)
	SVFDT-II	0.990	0.979	312	197.425 (0.583)
sea	VFDT	0.850	0.598	273	6.409 (0.022)
	SVFDT-I	0.852	0.603	116	6.438 (0.019)
	SVFDT-II	0.851	0.601	137	6.681 (0.019)
spam	VFDT	0.807	0.252	20	182.019 (1.708)
	SVFDT-I	0.768	0.102	8	174.086 (1.173)
	SVFDT-II	0.768	0.102	8	173.826 (1.938)
usenet	VFDT	0.547	0.086	33	3.819 (0.163)
	SVFDT-I	0.563	0.119	11	3.787 (0.047)
	SVFDT-II	0.561	0.114	9	3.795 (0.096)

for $\tau = 0.05$.

Table 4. Mean relative metrics against the VFDT for each tiebreak value.

τ	Algorithm	Rel. ACC	Rel. Kappa M	Rel. size	Rel. time
0.05	SVFDT-I	0.992	0.991	0.484	0.853
	SVFDT-II	1.000	1.006	0.669	0.971
0.10	SVFDT-I	0.984	0.844	0.412	0.877
	SVFDT-II	0.984	0.825	0.566	1.043
0.15	SVFDT-I	0.992	1.020	0.347	1.044
	SVFDT-II	0.997	1.043	0.529	1.125
0.20	SVFDT-I	0.999	1.076	0.353	1.103
	SVFDT-II	1.002	1.084	0.504	1.148

The statistical significance of the difference in accuracy, memory and training time were assessed using the Friedman's statistical test and the post-hoc test of Nemenyi. A Critical Difference (CD) diagram is used to illustrate the results from these tests. Figures 2, 3 and 4 present the CD diagrams for the accuracy, memory consumption and training time, respectively, using 95% of significance. They were constructed using the predictive performance of the trees with $\tau = 0.05$. According to the statistical tests, there were no statistically significant differences between VFDT predictive performance and the predictive performance of the two proposed algorithms. However, for memory used, there is a statistically significant difference only between SVFDT-I and VFDT, as shown in Figure 3. When considering training time, there was statistical difference between SVFDT-I and VFDT, but not for SVFDT-II and VFDT. Thus, for the datasets used in this study, SVFDT-I with $\tau = 0.05$ would be a better choice than VFDT, since it significantly reduced training time and memory, keeping a similar predictive performance.

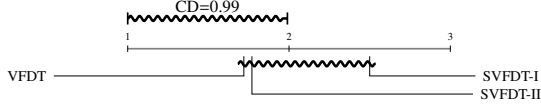The predictive performance of an algorithm on data stream

Fig. 2. Accuracy performance comparison among VFDT, SVFDT-I and SVFDT-II according to the Friedman and Nemenyi test using $\tau = 0.05$. There are no significantly different algorithms
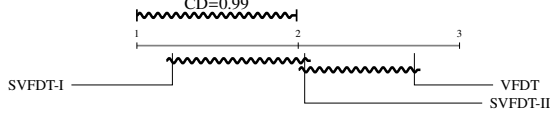

Fig. 3. Memory performance comparison among VFDT, SVFDT-I and SVFDT-II according to the Friedman and Nemenyi test using $\tau = 0.05$. Algorithms that are not significantly different are connected
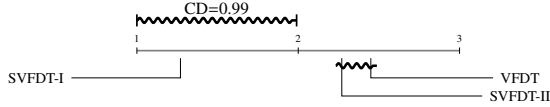

Fig. 4. Time performance comparison among VFDT, SVFDT-I and SVFDT-II according to the Friedman and Nemenyi test using $\tau = 0.05$. Algorithms that are not significantly different are connected

classification can be evaluated by looking at its performance along the stream (Gama et al., 2003). Accordingly, Figure 5 presents the accuracy and tree size, in number of nodes, during training, per dataset. It is possible to see that, as more instances were processed, both SVFDT algorithms kept predictive performance similar to VFDT. But when considering the size of the trees, both SVFDTs outperformed VFDT by a large margin in most datasets. It is also possible to notice periods where SVFDTs completely stops growing, while VFDT continues to grow, indicating that during these periods there is no need to increase the model size. This pattern can be observed in all datasets.

Datasets with concept drift and noise were analysed. Concept drifts are present in the sea, spam and usenet datasets. SVFDTs and VFDT predictive performance in the presence of concept drifts were similar, except in the dataset sea, when SVFDTs' predictive performance was better, using less than half of the memory used by VFDT.

Performance in the presence of noise was explored in the led dataset, more specifically, led24_10 and led24_20 (Figure 5). In these datasets, the SVFDTs were still able to significantly reduce the tree size regarding VFDT.

6. Conclusion and Future Work

This work proposed and experimentally investigated two versions of a new VFDT-based algorithm, SVFDT-I and SVFDT-II. The SVFDTs were created to reduce the size of the trees induced by VFDT, inducing a memory conservative decision tree for data stream mining. According to experimental results, Both SVFDTs induce trees significantly smaller than those induced by the VFDT, while not statistically compromising predictive performance. This study also assessed the influence of SVFDT-I hyperparameter τ value in the training time and size of induced trees. For all τ values investigated, on average, trees

at least 52% smaller with accuracy at most 1.6% lower than VFDT were created. Sometimes this came with an increase in training time, showing that "there is no free lunch". Finally, for almost all datasets, SVFDT-II presented higher predictive accuracy than the SVFDT-I, together with significantly reducing tree size. A statistical analysis of the performances of SVFDTs, when compared with VFDT, for $\tau = 0.05$, the value with the best results, showed no statistically significant difference in predictive performance, but a significant lower memory use and training time for SVFDT-I. These results show that SVFDTs can be an efficient alternative to the VFDT in data stream mining applications. As future work, we intend to investigate how to combine the proposed algorithms in ensembles, to increase predictive accuracy keeping low memory use and training time.

References

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

Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B., 2010. MOA: massive online analysis. Journal of Machine Learning Research 11, 1601–1604.

Breiman, L., 2001. Random forests. Machine Learning 45, 5–32.

Domingos, P., Hulten, G., 2000. Mining high-speed data streams, pp. 71–80.

Gama, J., Rodrigues, P.P., Spinosa, E., Carvalho, A., 2010. 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 , 125–138.

Gama, J.a., Rocha, R., Medas, P., 2003. Accurate decision trees for mining high-speed data streams, in: Proc. of the IX ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA. pp. 523–528.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The weka data mining software: An update. SIGKDD Explor. Newsl. 11, 10–18.

Holmes, G., Richard, K., Pfahringer, B., 2005. Tie-breaking in Hoeffding trees, in: Proc. of the II Int. Workshop on Knowledge Discovery from Data Streams.

Hulten, G., Spencer, L., Domingos, P., 2001. Mining time-changing data streams, in: Proc. of the VII ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01, pp. 97–106.

Katakis, I., Tsoumakas, G., Vlahavas, I., 2010. Tracking recurring contexts using ensemble classifiers: An application to email filtering. Knowledge and Information Systems 22, 371–391.

Krawczyk, B., Minku, L., Gama, J., Stefanowski, J., 2017. Ensemble learning for data stream analysis: A survey. Information Fusion 37, 1–86.

Pfahringer, B., Holmes, G., Kirkby, R., 2007. New Options for Hoeffding Trees. Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 90–99.

Pfahringer, B., Holmes, G., Kirkby, R., 2008. Handling numeric attributes in hoeffding trees, in: Proc. of the XII Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, pp. 296–307.

Street, W.N., Kim, Y., 2001. A streaming ensemble algorithm (SEA) for large-scale classification, in: Proc. of the VII ACM SIGKDD int. conference on Knowledge discovery and data mining - KDD '01, pp. 377–382.

Yang, H., Fong, S., 2011a. Optimized very fast decision tree with balanced classification accuracy and compact tree size. volume xxx. pp. 57–64.

Yang, H., Fong, S., 2011b. OVFDT with functional tree leaf - majority class, naive bayes and adaptive hybrid integrations, in: Proc. of thehe III International Conference on Data Mining and Intelligent Information Technology Applications, pp. 65–70.

Yang, H., Fong, S., 2013. Incremental optimization mechanism for constructing a decision tree in data stream mining. Mathematical Problems in Engineering 2013, 114–144.
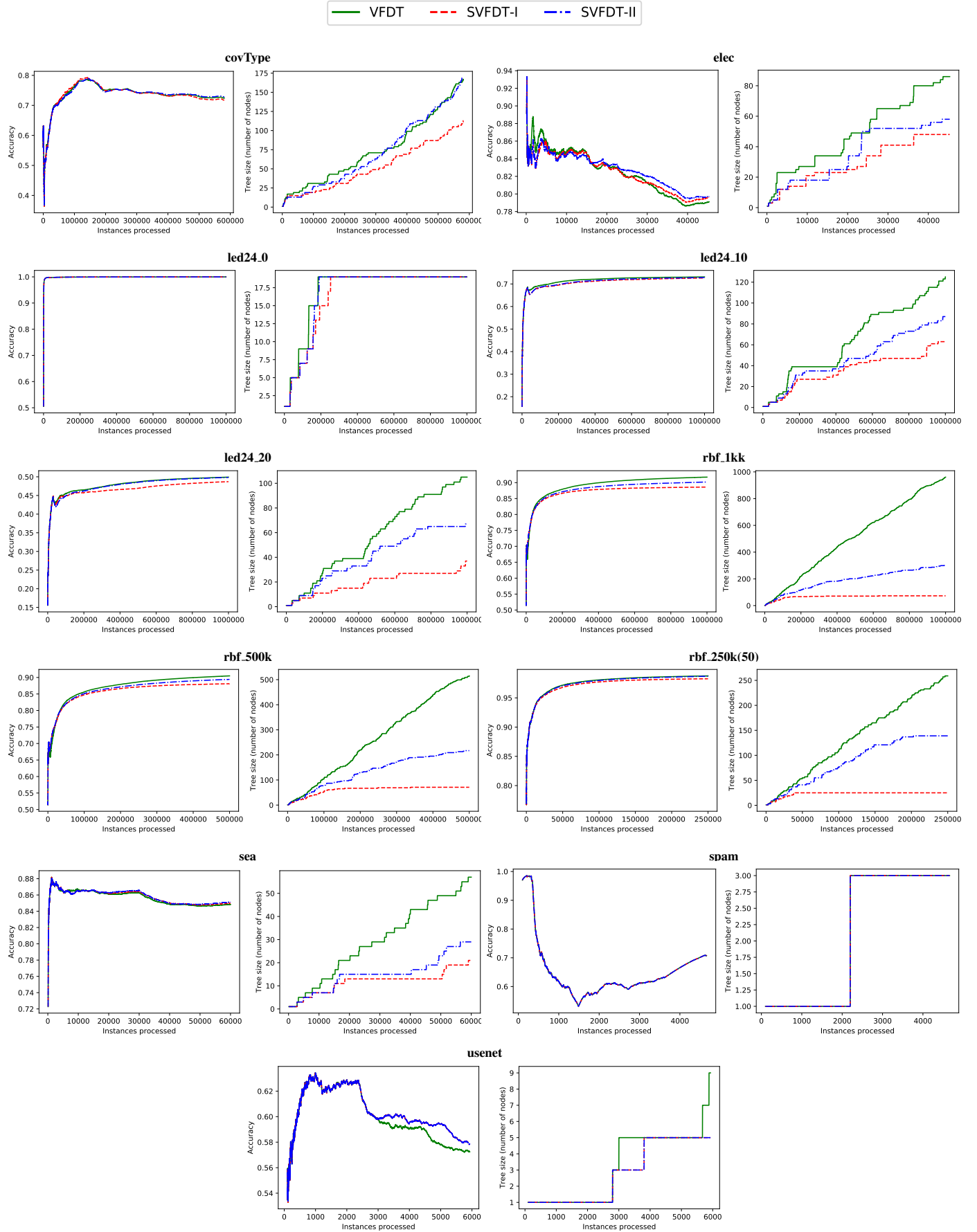
Fig. 5. Accuracy and tree size (in number of nodes) for training/testing with a tiebreak of 0.05