Speeding up Very Fast Decision Tree with Low Computational Cost

Jian Sun\(^1\), Hongyu Jia\(^1\), Bo Hu\(^2\), Xiao Huang\(^1\), Hao Zhang\(^1\), Hai Wan\(^1\)*, Xibin Zhao\(^1\)

\(^1\) KLISS, BNRist, School of Software, Tsinghua University, China
\(^2\) Beijing University of Posts and Telecommunications, China

\{sunj17, jiahy19\}@mails.tsinghua.edu.cn, huboniccolo@bupt.edu.cn,
\{huangx19, hao-zhan17\}@mails.tsinghua.edu.cn, \{wanhai, zxb\}@tsinghua.edu.cn

Abstract

Very Fast Decision Tree (VFDT) is one of the most widely used online decision tree induction algorithms, and it provides high classification accuracy with theoretical guarantees. In VFDT, the split-attempt operation is essential for leaf-split. It is computation-intensive since it computes the heuristic measure of all attributes of a leaf. To reduce split-attempts, VFDT tries to split at constant intervals (for example, every 200 examples). However, this mechanism introduces split-delay for split can only happen at fixed intervals, which slows down the growth of VFDT and finally lowers accuracy. To address this problem, we first devise an online incremental algorithm that computes the heuristic measure of an attribute with a much lower computational cost. Then a subset of attributes is carefully selected to find a potential split timing using this algorithm. A split-attempt will be carried out once the timing is verified. By the whole process, computational cost and split-delay are lowered significantly. Comprehensive experiments are conducted using multiple synthetic and real datasets. Compared with state-of-the-art algorithms, our method reduces split-attempts by about 5 to 10 times on average with much lower split-delay, which makes our algorithm run faster and more accurate.

1 Introduction

In recent years, the amount of data generated by industrial areas is growing significantly. [Dobre and Xhafa, 2014] pointed out that approximately 25 exabytes of data are generated every day. Traditional offline methods have to store all the data first, then apply specific algorithms to train models. However, if large amounts of data arrive continuously, due to the limited computing resources, the memory becomes a bottleneck. Training offline models after storing all data becomes increasingly impractical. Therefore, it is particularly important to design online algorithms to process data in real-time.

Over the past few decades, decision tree algorithms have been deeply studied and applied in various fields due to its high efficiency, accuracy, and interpretability. Traditional decision tree algorithms (ID3 [Quinlan, 1986], CART [Breiman et al., 1984], C4.5 [Quinlan, 1993], etc.) are not suitable for streaming environments. When a tree grows, the best attribute is first selected based on all the data, and then apply it to guide the tree split for learning more information. However, it is impossible to store all data of a stream, thus decision trees need to determine how much data should be accumulated to select the right attribute. VFDT [ Domingos and Hulten, 2000] and its variants (EFDT [ Manapragada et al., 2018 ], VFTDc [ Gama et al., 2003 ], etc.) are currently the most popular online decision tree methods, using Hoeffding bound as the theoretical guarantee to assume that a small number of samples are sufficient to select the best split attribute. By setting the parameter \( \delta \) in Hoeffding bound, VFDT can ensure that the attribute selected during splitting is the best attribute with 1 – \( \delta \) confidence.

In specific implementation details, since the volume of data needed to be accumulated cannot be predicted in advance, a naive approach is to perform a split-attempt of a leaf for every newly arrived example to check whether the best attribute selected from the current data satisfies the Hoeffding bound. This approach results in high computational cost, as each split-attempt needs to calculate heuristic measure functions (i.e., information gain [Quinlan, 1993] or Gini index [Breiman et al., 1984]) of all attributes. A large number of algorithms based on VFDT [Hulten et al., 2001; Gama et al., 2003; Bifet and Gavaldà, 2009; Ikonomovska et al., 2011a; Ikonomovska et al., 2011b] use the same approach to alleviate this problem by setting a hyperparameter \( n_{\min} \) and performs split-attempt every \( n_{\min} \) examples, therefore reducing split-attempts and shortening the running time. However, when \( n_{\min} \) is too large, the period of split-attempt is prolonged and it is more likely to miss the optimal splitting time, thereby inducing long split-delay which makes the tree grow slower and reduce accuracy. Conversely, small \( n_{\min} \) will increase runtime.

How to make the leaf split reasonably and fast without affecting the operational efficiency of the algorithm is the focus of this paper. Two main works of this paper are as follows:

1. A mechanism is proposed to incrementally update the heuristic measure of an attribute in constant time. By this mechanism, the heuristic measure can be updated immediately every time a single example arrives, instead
of being calculated periodically like VFD T.

2. We introduce the candidate attribute set which keeps track of the top K attributes that are most likely to split. Only the heuristic measures of attributes in the set are updated continuously, which further reduces the computational cost. To cover a wider range of attributes and make the set indeed save the true top K attributes, we propose an adaptive candidate set updating mechanism that dynamically changes candidate attributes when their performance deteriorates.

We evaluate and verify the algorithm on multiple synthetic and real datasets. Compared with the existing methods, our method can effectively reduce split-attempts by 5 to 10 times on average and even hundreds of times in some data streams, which greatly reduces the computational cost. What’s more, we shorten split-delay more stably in both stationary and non-stationary environments. These make our algorithm run faster and more accurate.

The rest of this paper is organized as follows: Section 2 introduces the related work. We describe our algorithm and the corresponding theoretical derivation in Section 3. Section 4 shows the experimental results and analysis. Finally, we conclude this paper in Section 5.

2 Related Work

Traditional decision tree algorithms are based on batch data, but [Domingos and Hulten, 2000] proposes VFD T for establishing decision trees using stream data based on Hoeffding bound [Hoeffding, 1994]. Suppose we have n independent observations of real-valued random variable r with range R and mean $\bar{r}$. The Hoeffding bound states that the true mean of the variable r is at least $\bar{r} - \epsilon$ with probability $1 - \delta$, where,

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

Denote $G(X_i)$ as the heuristic measure of attribute $X_i$ at the leaf node. Assume that after observing n pieces of data, $X_a$ and $X_b$ are the attributes with highest and second-highest G. Let $\Delta G = G(X_a) - G(X_b)$ and if $\Delta G > \epsilon$, then the attribute $X_a$ has a probability of $1 - \delta$ as the best attribute of the current leaf node. Since the calculation of $\Delta G$ needs to be performed on all attributes, VFD T uses the hyperparameter $n_{min}$ to periodically check if $\Delta G > \epsilon$ satisfied. The introduction of $n_{min}$ effectively improves the runtime of VFD T. But fixed $n_{min}$ will also delay the split and affect the performance of VFD T.

In recent studies, more and more people focus on accelerating the learning of VFD T, because the Hoeffding bound is still a conservative measure [Das et al., 2019]. [Manapragada et al., 2018] proposed EF DT (Extremely Fast Decision Tree) which uses a more loose judgment to speed up the splitting of leaves. If $\Delta G = G(X_a) - G(X_b) > \epsilon$, which means splitting on the best attribute works better than not splitting, then EF DT will split attribute $X_a$ at the leaf node. However, this method requires a periodic check because the best attributes are prone to change. To accelerate the learning process, Mem-ES [Das et al., 2019] abandons the theoretical guarantee of Hoeffding’s inequality. Based on the principle of Bag of Little Bootstraps [Kleiner et al., 2012], Mem-ES employs a sampling method to speed up splitting. But this method retains some examples at leaves, which lead to large memory consumption.

Scholars have begun to pay attention to the hyperparameter $n_{min}$ in the past few years, [García-Martín, 2017] assumes that the $\Delta G$ value calculated for the first time does not change. As $\epsilon$ in Equation 1 decreases with the number of samples n increasing, it estimates that at least how many samples are required to be observed in order to make $\epsilon$ small enough to satisfy the inequality $\Delta G > \epsilon$. [Losing et al., 2018] proposes the OSM algorithm, which assumes that $G(X_b)$, the heuristic measure of the second-best attribute, does not change, the value of $G(X_a)$ increases as much as possible as the data arrives. In this case, OSM estimates how many samples are needed to satisfy the Hoeffding bound, such that $G(X_a) - G(X_b) > \epsilon$. Obviously, the value of $\Delta G$ or $G(X_b)$ actually changes dynamically. Although these approaches may effectively reduce split-attempts, they will increase the split-delay and have a negative impact on the accuracy of the algorithm.

Compared with previous algorithms, the algorithm proposed in this paper can use both the historical and the latest data distribution information, generate fewer split-attempts and less split-delay, it can also improve the accuracy to a certain extent. Besides, our algorithm can be well embedded in the above algorithms such as VFD T, EF DT and so on.

3 Methodology

3.1 Preliminary

Algorithm 1 Online Decision Tree Induction

Input: S: A sequence of examples; G: A heuristic measure function; X: A set of attributes; $\Delta$: One minus the desired probability of choosing the correct attribute at any given node; $\tau$: The threshold of tie-split; $n_{min}$: The period of split-attempt.

Output: HT: Online decision tree learned from S

1: Initialize HT with a single root node
2: Initialize the statistics for tree growth
3: for each example s in S do
4: Sort s into a leaf l using HT
5: Update the statistics at l for tree growth
6: if the weight of examples at $l \bmod n_{min}$ equals 0 then
7: AttemptToSplit(l, G, $\tilde{X}$, $\Delta$, $\tau$)
8: end if
9: end for
10: return HT

Algorithm 1 Online Decision Tree Induction

1: Initialize HT with a single root node
2: Initialize the statistics for tree growth
3: for each example s in S do
4: Sort s into a leaf l using HT
5: Update the statistics at l for tree growth
6: if the weight of examples at $l \bmod n_{min}$ equals 0 then
7: AttemptToSplit(l, G, $\tilde{X}$, $\Delta$, $\tau$)
8: end if
9: end for
10: return HT
nodes is log (n + 1) G
best attribute q attributes. Assuming that the current node can be split into we further extend it to the heuristic measure calculation of G
when two attributes have very similar G values, if we only use the condition G(Xa) − G(Xb) > ε, potentially many examples will be required to split a leaf. VFDT uses condition ε < τ to determine whether the tie situation occurs. It uses the current best attribute to split when the tie situation does occur. The split in this situation is called tie-split. The parameter τ essentially specifies the maximum number of examples that can be accumulated at a leaf.

3.2 Incremental Heuristic Measure
[Sovdat, 2014] proposes the updating formulas of computing entropy and Gini index for time-changing data streams, and we further extend it to the heuristic measure calculation of attributes. Assuming that the current node can be split into q child nodes using attribute Xi. The data set in the current node is D, and the number of classes is c, then the following symbols can be defined:

- n: the total weight of examples in D;
- ni,j: the total weight of examples that would be passed into the jth child node if the split was made by the attribute Xi;
- ni,j,k: the total weight of examples from the kth class that would be passed into the jth child node if the split was made by the attribute Xi.

For the clarity of the explanation, the index i will be neglected, i.e., ni,j ≡ nj, ni,j,k ≡ nj,k.

If VFDT uses information gain as the heuristic measure, the conditional entropy on the attribute Xi is:

\[ G(D|X_i) = -\frac{1}{n} \sum_{j=1}^{q} \sum_{k=1}^{c} \frac{n_{i,j,k}}{n_j} \log \frac{n_{i,j,k}}{n_j} \] (2)

**Theorem 1.** For any given example s with weight w and class m (m ∈ [1, c]), suppose s will be passed into the rth child node if the split was made by the attribute Xi, then the new conditional entropy of the attribute Xi is \( G(D|X_i) = \frac{1}{n+1} [nG(D|X_i) − \Delta] \), where \( \Delta = \log \left( \frac{n_r}{n_r + w} \left( \frac{n_{r,m} + w}{n_{r,m}} \right) \right) \)

**Proof.** According to the assumptions of Theorem 1, both nr and nr,m will increase w, and the values of other parameters remain unchanged. Then

\[
G(D|X_i) = -\frac{n}{n+w}G(D|X_i)
= -\frac{1}{n+w}(n_{r,m} \log \frac{n_{r,m}+w}{n_{r,m}} + w \log \frac{n_{r,m}+w}{w+n})
- \frac{1}{n+w} \sum_{k=1}^{c} n_{r,k} \log \frac{n_{r,k}}{n_{r}+w}
= -\frac{1}{n+w} \log \left( \frac{n_r}{n_r+w} \left( \frac{n_{r,m}+w}{n_{r,m}} \right) \right) \left( \frac{n_{r,m}+w}{n_{r,m}} \right)^w 
\] (3)

This completes the demonstration of Theorem 1.

When VFDT uses Gini index as the heuristic measure, the term in which does not depend on the attribute, will be neglected since the subject of interest is the difference of the Gini index for two attributes. Thus the approximate Gini index of the attribute Xi is:

\[
G(D|X_i) = \frac{1}{n} \sum_{j=1}^{q} \sum_{k=1}^{c} \frac{n_{i,j,k}}{n_j} \] (4)

We can prove Theorem 2 in the same way as theorem 1.

**Theorem 2.** For any given example s with weight w and class m (m ∈ [1, c]), suppose s will be passed into the rth child node if the split was made by the attribute Xi, then the new Gini index of the attribute Xi is

\[
G(D|X_i) = \frac{1}{n+w} [nG(D|X_i) + \Delta], \quad \Delta = 2w(n_{r,m} + w^2) - \sum_{k=1}^{c} w(n_{r,k})^2
\]

3.3 Incremental Measure Algorithm Based on Candidate Attributes (IMAC)
According to Theorem 1 and Theorem 2, when the heuristic measure is information gain or Gini index, it can be incrementally calculated respectively. Based on this, an incremental algorithm is proposed, which we call IMAC. The algorithm is described in Algorithm 3 and Function 4.

IMAC maintains a candidate attribute set P in each leaf node, which stores the top K attributes that are most likely to split. In our algorithm, K is equal to 10% of the number of attributes. To avoid too many or too few candidate attributes are selected, the value of K is limited between 5 to 10. As long as P is not empty, the heuristic measure of the current candidate attributes will be incrementally updated according to Theorem 1 or 2 (lines: 6-8). When the difference between the heuristic measure of the best attribute and the second-best attribute in P is greater than the Hoeffding bound, the split-attempt operation will be executed at the leaf (lines: 10-13).

In function 4, if the split-attempt operation fails, IMAC will replace P with new top K attributes according to the rank of current values of the heuristic measure (line: 9).
Algorithm 3 Online Decision Tree Induction with IMAC

Input: $S$: A sequence of examples; $G$: A heuristic measure function; $X$: A set of attributes; $\delta$: One minus the desired probability of choosing the correct attribute at any given code; $\tau$: The threshold of tie-split; $\eta$: The minimum total weight of examples must be accumulated at a leaf; $K$: The maximum size of the candidate attribute set.

Output: $HT$: Online decision tree learned from $S$

1. Initialize $HT$ with a single root node
2. Initialize the statistics for tree growth and an empty candidate attribute set $P$
3. for each example $s$ in $S$ do
4.     Sort $s$ into a leaf $l$ using $HT$
5.     Update the statistics at $l$ for tree growth
6. if candidate attribute set $P_l$ at $l$ is not empty then
7.     for each attribute $X_i \in P_l$ do
8.         Compute the heuristic measure of $X_i$ using Theorem 1 or 2
9.     end for
10. $X_a, X_b$ are the two attributes of $P_l$ with highest $G_l$
11. Compute $\epsilon$ using Equation 1
12. if $G(X_a) - G(X_b) > \epsilon$ or $\epsilon < \tau$ then
13.     AttemptToSplitWithIMAC($l, G, X, \delta, \tau, K$)
14. end if
15. else if the weight of examples at $l$ is greater than $\eta$ then
16.     AttemptToSplitWithIMAC($l, G, X, \delta, \tau, K$)
17. end if
18. end for
19. return $HT$

Function 4 AttemptToSplitWithIMAC($l, G, X, \delta, \tau, K$)

1. $X_a, X_b$ are the two attributes with highest $G_l$
2. Compute $\epsilon$ using Equation 1
3. if $G(X_a) - G(X_b) > \epsilon$ or $\epsilon < \tau$ then
4.     Replace $l$ with an internal node
5.     for each branch of the split do
6.         Add a new leaf $l_m$ with empty $P_m$ and let $X_m = X - \{X_a\}$
7.     end for
8. else
9.     Replace $P_l$ with new top $K$ attributes according to the rank of $G_l$
10. end if

3.4 Adaptive Switching Mechanism for Candidate Attributes

Theoretically, the result based on theorem 1 or theorem 2 should be accurate. However, the following situations may occur, resulting in an inaccurate result:

- In some improved algorithms based on VFDT, to speed up the processing of numerical attributes, it is assumed that the distribution of the values of a numerical attribute follows a normal distribution, and only the mean and variance of numerical attributes are stored in a leaf node [Gama et al., 2004]. When calculating the heuristic measure of a numerical attribute, the optimal split-point of the attribute needs to be considered additionally. At a given split-point, the approximate weight greater than or less than the split-point is estimated by the normal distribution. While in the incremental method of IMAC, the weight is calculated accurately at a given split-point. The difference between the incremental method and the approximation method on a numerical attribute will lead to a large deviation finally.

- The candidate attribute set $P$ may not have captured the true split attribute. On the one hand, due to the small amount of data at the beginning, the difference of the heuristic measure between the true best attribute and other attributes is not obvious, and even the true best attribute may be smaller. If $P$ is initialized at this time point, the true best attribute may not be included. On the other hand, if concept drift occurs, the data distribution can change over time [Schlimmer and Granger, 1986; Widmer and Kubat, 1996]. Thus the best attribute may change at different stages, which is difficult to capture. For numerical attributes, even if the best attribute does not change, the optimal split-point may be different at different stages by calculating the data weight approximately.

To solve these problems, we give the following rules:

**Illegal Attribute.** Assume $X_a$ is the attribute with highest $G$. For any other attribute $X_i, X_i$ is said to be illegal if $G(X_a) - G(X_i) > \epsilon$. We believe that there is sufficient evidence to show that $X_a$ is better than $X_i$, $X_i$ will not be considered anymore and should be added into the illegal attribute set $I$.

**Potential Attribute.** Assume $X_a$ is the attribute with highest $G$. For any other attribute $X_i, X_i$ is said to be potential if $G(X_a) - G(X_i) > \epsilon$ and $X_i$ is not in the candidate set $P$, then $X_i$ should be added into the potential attribute set $M$.

A hyperparameter $\mu$ is used to check the candidate set $P$ every $\mu$ examples. Assume $X_p$ is the worst attribute with lowest $G$ in $P$, and $X_a$ is the best attribute with highest $G$ in $M$. Since the attribute information in $M$ is not updated incrementally in time, the latest value of the heuristic measure of $X_a$ needs to be recalculated. The one with the larger $G$ of the two attributes will be added to $P$ and the other will be added to $I$ (according to the rule of illegal attribute) or $M$ (according to the rule of potential attribute).

**Space and Time Complexity Analysis**

If $c$ is the number of classes, $d$ is the number of attributes and each nominal attribute can have at most $v$ values, then VFDT maintains the statistics of all attributes at each leaf in $O(dvc)$ memory. Since IMAC introduces $P$ to save the incremental information of top $K$ candidate attributes, it requires extra $O(Kvc)(K < d)$ memory. $M$ and $I$ only store the names and latest $G$ values of attributes, the memory required is $O(d)$. In conclusion, the total required memory of each leaf in IMAC is still $O(dvc)$.

The time to calculate the heuristic measure of all attributes in VFDT is $O(dvc)$. Assume that the total weight of the data observed at a leaf is $n$, $n_v = n/n_{min}$ split-attempts are required on average in VFDT, and $n_i$ split-attempts are re-
required in IMAC. Thus the time complexity of split-attempts is $O(n_d v c)$ in VFDT. Except for split-attempt operation in IMAC, $P$ needs to be checked periodically in $O(n_c K_v c)$ ($n_c = n/\mu$) time, the time of candidate attributes are updated incrementally in $O(n K)$ time. Thus the time of split-attempt and the time of additional operations introduced in IMAC is $O(n_d v c + n_c K_v c + n K)$. If $\frac{n_c}{n_d} < 1 - \frac{K_{n_{\text{min}}}}{\mu d} - \frac{K_{n_{\text{min}}}}{v c}$, IMAC will faster than VFDT. We define a new function $\varphi = 1 - \frac{K_{n_{\text{min}}}}{\mu d} - \frac{K_{n_{\text{min}}}}{v c}$. $\varphi$ is an increasing function with $d$, $v$ and $c$ as arguments. From this, we know that, in large-scale and complex data flow scenarios, IMAC is very likely to be faster than VFDT. In Section 4, IMAC usually reduces split-attempts by about 10 times compared with VFDT, and sometimes even hundreds of times. In industrial scenarios, data is usually very complex, with hundreds of attributes, and some attributes may have hundreds of different state values. IMAC is more suitable for these complex scenarios.

4 Experiment

4.1 Experimental Setup

Environment

All algorithms and experiments are implemented on the Massive Online Analysis (MOA) platform [Bifet et al., 2010], which is one of the most popular open-source frameworks for data stream mining. Our optimizations can also be integrated on other platforms (i.e., STREAMDM C++ [Bifet et al., 2017], Scikit-Multiflow [Montiel et al., 2018]). Our code is available at GitHub1. All experiments are conducted on a standard server with 36 cores and 125GB memory. We evaluate the performance of three methods in this paper:

- The standard VFDT with periodic split-attempts.
- The current best algorithm OSM, which predicts the interval of split-attempts in VFDT and has the best performance optimization effect [Losing et al., 2018].
- The algorithm IMAC in this paper, which determines the potential split timing in VFDT with incremental information.

VFDT with default parameters ($n_{\text{min}} = 200$, $\tau = 0.05$, $\delta = 1e - 7$), uses the majority class in leaves for classification and information gain as the heuristic measure. Since $n_{\text{min}}$ is 200, to compare with VFDT and OSM at the same level, parameter $\mu$ and $\eta$ in IMAC are both set to 200.

Datasets

We use large streams consisting of well known real-world and synthetic datasets. Table 1 shows detailed information.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Samples</th>
<th>#Feat</th>
<th>#Class</th>
<th>Type</th>
<th>Stationarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEA$_g$</td>
<td>5M</td>
<td>3</td>
<td>2</td>
<td>artificial</td>
<td>No</td>
</tr>
<tr>
<td>LED$_g$</td>
<td>1M</td>
<td>24</td>
<td>10</td>
<td>artificial</td>
<td>No</td>
</tr>
<tr>
<td>AGR$_g$</td>
<td>10M</td>
<td>9</td>
<td>2</td>
<td>artificial</td>
<td>No</td>
</tr>
<tr>
<td>RBF</td>
<td>2M</td>
<td>200</td>
<td>15</td>
<td>artificial</td>
<td>Yes</td>
</tr>
<tr>
<td>RTG</td>
<td>10M</td>
<td>400</td>
<td>25</td>
<td>artificial</td>
<td>Yes</td>
</tr>
<tr>
<td>Covertype</td>
<td>581K</td>
<td>54</td>
<td>7</td>
<td>real</td>
<td>No</td>
</tr>
<tr>
<td>KDD99</td>
<td>4.9M</td>
<td>41</td>
<td>23</td>
<td>real</td>
<td>No</td>
</tr>
<tr>
<td>MNIST8M</td>
<td>8.1M</td>
<td>784</td>
<td>10</td>
<td>real</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1: Datasets information.

By comparison to VFDT, although OSM usually reduces split-attempts by 2 to 8 times, the performance of split-delay is particularly unstable. OSM can maintain roughly the same performance in others, sometimes even more than 10 times (for example, RBF, Covertype and KDD99). The reason is that OSM reduces split-attempts and substantially reduces it by more than 4 times normally. Although fewer split-attempts are used, less split-delay is introduced, which makes IMAC

1https://github.com/yearsj/IMAC
Table 2: A comprehensive evaluation of VFDT, OSM and IMAC, including split-attempts, the average of split-delay, total time and accuracy.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#Attempts</th>
<th>Average delay</th>
<th>Total Time (s)</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VFDT</td>
<td>OSM</td>
<td>IMAC</td>
<td>VFDT</td>
</tr>
<tr>
<td>SEA$_d$</td>
<td>25436</td>
<td>18172</td>
<td>3414</td>
<td>96.95</td>
</tr>
<tr>
<td>LED$_d$</td>
<td>4999</td>
<td>2652</td>
<td>186</td>
<td>113.80</td>
</tr>
<tr>
<td>AGR$_d$</td>
<td>33229</td>
<td>20496</td>
<td>9688</td>
<td>98.62</td>
</tr>
<tr>
<td>RBF</td>
<td>10008</td>
<td>1649</td>
<td>105</td>
<td>121.00</td>
</tr>
<tr>
<td>RTG</td>
<td>35949</td>
<td>9565</td>
<td>2499</td>
<td>100.34</td>
</tr>
<tr>
<td>Covertype</td>
<td>2853</td>
<td>1237</td>
<td>492</td>
<td>99.43</td>
</tr>
<tr>
<td>KDD99</td>
<td>4231</td>
<td>105</td>
<td>113</td>
<td>102.50</td>
</tr>
<tr>
<td>MNIST8M</td>
<td>40436</td>
<td>11885</td>
<td>527</td>
<td>68.40</td>
</tr>
</tbody>
</table>

Figure 1: As the increase of #attribute and #examples, the variation of total-time and attempt-time on different algorithms in RTG. All time values are processed by logarithmic base 2.

5 Conclusion

In this paper, we further improved the performance of VFDT by replacing its periodic split-attempt mechanism. We calculated the heuristic measure incrementally and applied it to determine the potential split-time on candidate attributes. To make the candidate set to cover a wider range, we also proposed a dynamic candidate set switching mechanism. We conducted a comprehensive experiment on multiple synthetic datasets and real datasets. Compared with state-of-the-art algorithms, IMAC can not only reduce split-delay but also significantly reduce split-attempts, which makes IMAC run faster and more accurate. Moreover, we verified that IMAC is more efficient in large-scale data streams.

Acknowledgments

This research is sponsored in part by the NSFC Program (No.U1801263) and the Key-Area Research and Development Program of Guangdong Province (No.2020B010164001).
References


