Streaming Decision Trees and Forests

Haoyin Xu[1,*], Jayanta Dey[1], Sambit Panda[1], and Joshua T. Vogelstein[1]

Abstract. Machine learning has successfully leveraged modern data and provided computational solutions to innumerable real-world problems, including physical and biomedical discoveries. Currently, estimators could handle both scenarios with all samples available and situations requiring continuous updates. However, there is still room for improvement on streaming algorithms based on batch decision trees and random forests, which are the leading methods in batch data tasks. In this paper, we explore the simplest partial fitting algorithm to extend batch trees and test our models: stream decision tree (SDT) and stream decision forest (SDF) on three classification tasks of varying complexities. For reference, both existing streaming trees (Hoeffding trees and Mondrian forests) and batch estimators are included in the experiments. In all three tasks, SDF consistently produces high accuracy, whereas existing estimators encounter space restraints and accuracy fluctuations. Thus, our streaming trees and forests show great potential for further improvements, which are good candidates for solving problems like distribution drift and transfer learning.

1 Introduction

In recent decades, machine learning methods continue to facilitate the utilization of modern data and make scientific progress in health care, technology, commerce, and more [1]. Their applications span across both fields with all samples ready to use and those with continuous data inputs, especially online data mining [2]. Among all the methods, batch decision trees (DTs) and ensembles like random forests (RFs) are the leading strategies for batch tasks like classification, outperforming all others on real datasets and machine learning competitions [3–6]. However, there lacks state-of-the-art streaming trees and forests on tasks having large sample sizes and requiring continual partial fitting. Fitting datasets with larger sample sizes requires overwhelmingly more amounts of computational time and space. Moreover, different scenarios of streaming inputs would need flexible strategies for model updates [7]. This means that, for batch tree estimators like DT and RF, all sample data must be stored, and re-fitting everything is needed for each update [8, 9].

Even with enough computational resources to do so, out-of-distribution (OOD) problems could undermine the validity of older data [10]. In contrast, by partially fitting indefinite batches of new training samples, streaming trees can continuously update the tree structure without storing old data [2, 11–13]. Furthermore, in forest ensembles, older trees could potentially be pruned to keep up with the current data distribution. In this paper, we explore the simplest partial fitting method to extend a fitted decision tree and introduce our two streaming tree implementations: stream decision tree (SDT) and stream decision forest (SDF). We test these estimators on three classification tasks with varying complexities. For reference, existing streaming trees like Hoeffding tree (HT) and Mondrian forest (MF), and batch estimators (DT and RF) are also included [2, 12].

Among all streaming tree methods, our SDF produces consistently accurate results, whereas HT shows significant fluctuations in accuracy. MF, on the other hand, requires much more computational space and fails to perform consistently. Overall, SDT and SDF show good potentials for further improvements and would be better candidates for solving OOD problems [10]. We can also use them to enhance transfer learning by adding new data to previously learned tasks [14].

2 Methods

[1]Johns Hopkins University *Corresponding author: hxu36@jhu.edu

2.1 Algorithms Both SDT and SDF are based on a customized fork of scikit-learn, which added the new `partial_fit` function to `DecisionTreeClassifier` [15]. A SDT is initialized almost in the same way as a DT, where the tree fits the first training samples. The only difference is the input of predefined class labels (cls), as the first batch might not include all possible classes. After SDT is fitted, each arriving training set $\mathcal{D}_n$ would be sorted into leaf nodes marked as "false roots" (Algorithm 1). These "false roots" are then split and extended if the minimum sample size (two in default) is reached.

A SDF, defined as `StreamDecisionForest`[1], is initialized as a collection of SDTs (100 in default). As in Algorithm 2, it partially fits the trees by randomly permuting the training batches given and limiting the number of features selected per split ("max-features") to $\sqrt{d}$, where d is the number of features. The predictions are generated by majority voting [9, 16, 17]. We benchmarked both SDT and SDF by training them with incremental batches of 100 random samples.

Algorithm 1 Partially fit a SDT with one batch of training samples.

Input:
(1) $\mathcal{D}_n = (\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{n \times p} \times \{1, \ldots, K\}^n$ ▷ batch of training samples
(2) cls ▷ classes, required for first call
Output:
(1) T ▷ partially fitted tree

function SDT.PARTIAL_FIT($\mathbf{x}, \mathbf{y}, cls$)
  if tree is not fitted **then**
    return SDT.fit($\mathbf{x}, \mathbf{y}, cls$) ▷ same as batch tree fitting except preset classes
  end if
  for each (x,y) in ($\mathbf{x}, \mathbf{y}$) **do**
    find the leaf node x would be sorted into ▷ same as prediction
    mark the node as a "false root" and associate (x,y) with it
  end for
  for each "false root" **do**
    update the node with associated samples
    split the node if satisfying minimum sample size
  end for
  return T
end function

2.2 Reference Algorithms For comparison with existing streaming trees, we selected two popular algorithms: HT and MF, and benchmarked them also with incremental batches of 100 random samples [2, 12, 18]. Open source code was used: `HoeffdingTreeClassifier` from the river package (BSD-3-Clause) and `MondrianForestClassifier` from the scikit-garden package (BSD-3-Clause) [19, 20]. We set the max size of HT as 1,000 MB and used other hyperparameters as default. MF used the default 10 estimators to guarantee enough computational space.

For comparison with batch estimators, we included DT and RF: `DecisionTreeClassifier` and `RandomForestClassifier` from the original scikit-learn package (BSD-3-Clause) [15]. At each sample size, we trained both classifiers with all available samples, and all hyperparameters were kept as default.

[1] https://neurodata.io/sdtf/

Algorithm 2 Partially fit a SDF with one batch of training samples.

Input:
(1) $\mathcal{D}_n = (\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{n \times p} \times \{1, \ldots, K\}^n$ ▷ batch of training samples
(2) cls ▷ classes, required for first call
Output:
(1) $\mathbf{T}$ ▷ partially fitted forest

function SDF.PARTIAL_FIT($\mathbf{x}, \mathbf{y}, cls$)
  for each T in $\mathbf{T}$ **do**
    $(\mathbf{x}^r, \mathbf{y}^r) \leftarrow$ randomize $(\mathbf{x}, \mathbf{y})$
    $T \leftarrow T$.partial_fit($\mathbf{x}^r, \mathbf{y}^r, cls$)
  end for
  return $\mathbf{T}$
end function

2.3 Data We selected three datasets (Table 1) for the classification tasks: Splice-junction Gene Sequences (splice), Pen-Based Recognition of Handwritten Digits (pendigits), and CIFAR-10 [21–23]. For the splice dataset, we randomly reserved 25% of all genetic sequences for testing [24, 25]. Each sequence contains 60 nucleotide codes, which are converted into numerical features by ordinal encoding: $A = 0.25$, $C = 0.50$, $G = 0.75$ and $T = 1.00$. All ambiguous nucleotides are represented by zero [26]. This task tests MF's performance on DNA data without feature engineering [12]. The pendigits dataset contains handwritten images represented by 16 pixel features, which are generated through spatial resampling [27]. The CIFAR-10 dataset contains RGB-colored images, each high-dimensional image represented by $32 \times 32 \times 3 = 3,072$ pixel features. We used the provided training and test sets for pendigits and CIFAR-10, and ran all tasks 10 times by randomizing the training sets.

Name	No. features	No. classes	No. training samples	No. test samples
splice	60	3	2,392	798
pendigits	16	10	7,494	3,498
CIFAR-10	3,072	10	50,000	10,000

Table 1: Dataset attributes.

2.4 Evaluation Metrics Performance is evaluated by classification accuracy and training wall times. The training wall times calculate the fitting times for the given model without hyperparameter tuning. All estimators' fitting times are calculated by accumulating the training times of data batches, which account for batch estimators' refitting. We dot not report error bars as they are minimal. All experiments were performed without parallelization on the same Microsoft Azure compute instance: a 6-core Standard_NC6 (Intel Xeon E5-2690 v3) with 56 GB memory and 340 GB SSD storage.

3 Results

3.1 Accuracy SDF tends to achieve high accuracy across all sample sizes and datasets, as compared to MF and HT (Figure 1, upper row). It performs as good as or better than DT. HT has the lowest accuracy when training on the splice dataset, which almost remains constant as sample size increases. It also experiences significant fluctuations when training on the pendigits and CIFAR-10 datasets. We speculate that this issue is caused by insufficient memory allocation, but increasing the max size could

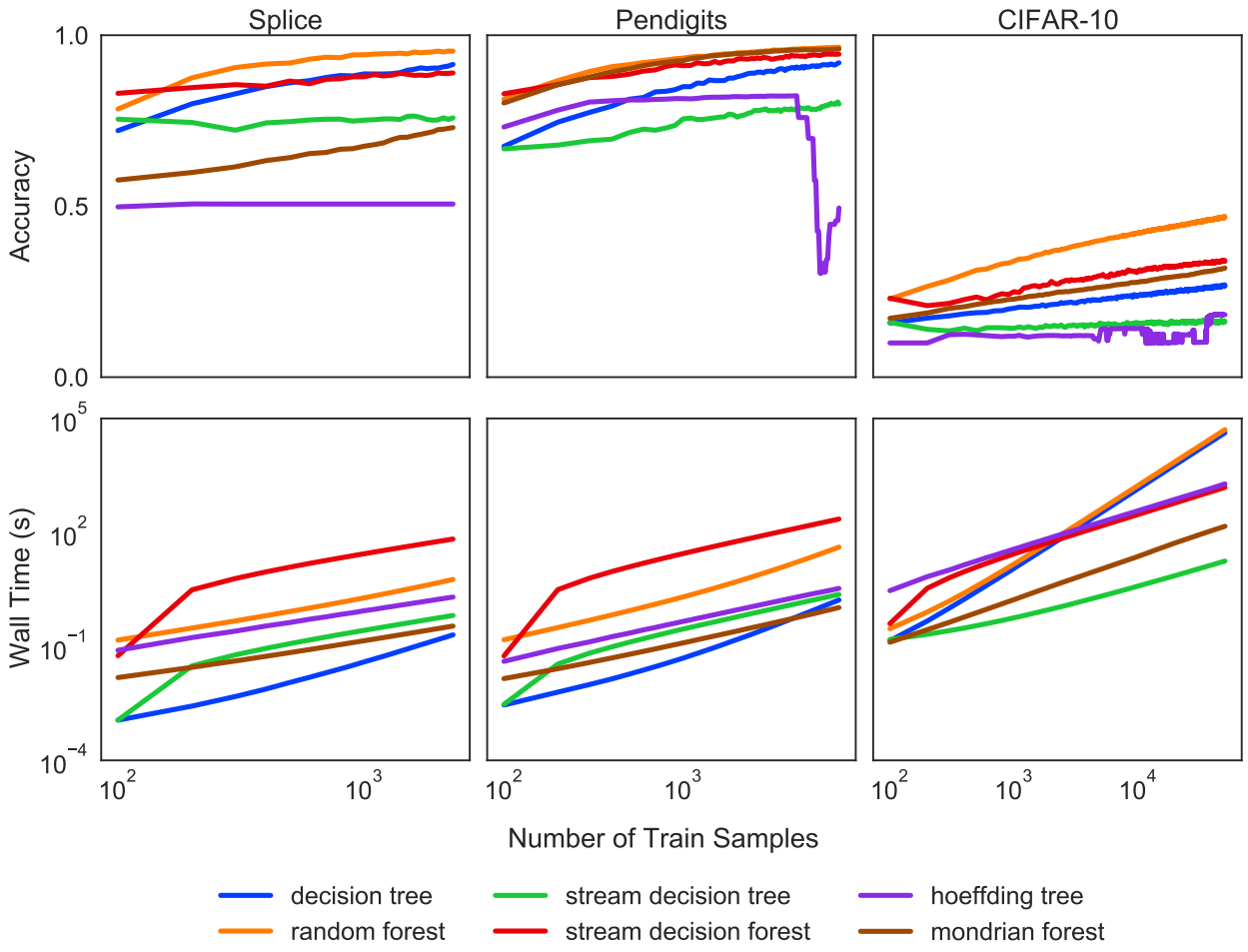

Figure 1: Multiclass classifications on splice **(left)**, pendigits **(center)**, and CIFAR-10 **(right)** datasets. Each line represents averaged results from 10 randomized repetitions. At larger sample sizes, stream decision forests (SDF) performs as well as or better than batch decision tree (DT) or other streaming estimators. Hoeffding tree (HT) either remains almost constant or experiences significant fluctuations in accuracy. Mondrian forest (MF) performs best in the pendigits task but only surpass HT in the splice task. Random forest (RF) achieves the highest accuracy in all tasks. For training wall times, cumulative fitting times are shown for all estimators, and two batch estimators' times include re-fitting at each sample size. SDT is the most efficient in the CIFAR-10 task. SDF takes longer times in the splice and pendigits tasks, but DT and RF overtake all streaming classifiers in the CIFAR-10 task. HT times increase significantly on the CIFAR-10 dataset.

not alleviate the problem. MF performs as good as RF in the pendigits task but only surpass HT on accuracy in the splice dataset. We find the difference as expected due to previous experiments, and the fluctuations could be attributed to the random partitions of Mondrian processes [12, 28]. The accuracy should be improved if feature engineering is implemented. RF achieves the highest accuracy at most sample sizes.

3.2 Training Wall Times SDT is the most efficient in the CIFAR-10 task. SDF takes longer training times in the two tasks with lower complexities, but DT and RF surpass all streaming estimators at larger sample sizes in the CIFAR-10 task (Figure 1, bottom row). The training times of HT increase significantly in the CIFAR-10 task, whereas MF's maintain more consistent trajectories.

4 Discussion Our implementations of streaming trees (SDT and SDF) utilize the simplest partial fitting strategy (Algorithm 1), and achieve consistent and competent results in the three classification tasks. Also, SDT shows the best time efficiency on the high-dimensional CIFAR-10 dataset. By comparison, under a reasonable computational environment, we find memory overflow issues and performance fluctuations on two existing streaming estimators: HT and MF. This shows our methods' advantages on accuracy, stability and efficiency.

Batch estimators could be more efficient and accurate if all samples are given at once, which would be nonetheless unlikely in real-world scenarios. In both our experiments and actual practices, batch decision trees and forests need to store all available data and re-fit them every time a new batch of samples comes in. Even by restricting the model updates with minimum batch samples, data storage would still cost significant computational resources.

Overall, SDT and SDF show good potentials for further improvements. By optimizing our approaches, we intend to construct robust streaming estimators and add streaming trees to the scikit-learn package [15]. These models would be good candidates for solving OOD problems, and transfer learning could leverage the streaming capability to update existing tasks.

References

[1] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. Science, 349(6245):255–260, 2015.

[2] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '00, pages 71–80, New York, NY, USA, 2000. Association for Computing Machinery.

[3] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd International Conference on Machine Learning, ICML '06, pages 161–168, New York, NY, USA, 2006. ACM.

[4] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In Proceedings of the 25th international conference on Machine learning, pages 96–103, New York, New York, USA, July 2008. ACM.

[5] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? Journal of Machine Learning Research, 15(90):3133–3181, 2014.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 785–794, New York, NY, USA, August 2016. Association for Computing Machinery.

[7] Hanady Abdulsalam, David B. Skillicorn, and Patrick Martin. Streaming random forests. In 11th International Database Engineering and Applications Symposium (IDEAS 2007), pages 225–232, 2007.

[8] Yali Amit and Donald Geman. Shape quantization and recognition with randomized trees. Neural Computation, 9(7):1545–1588, 07 1997.

[9] Leo Breiman. Random forests. Machine learning, 45(1):5–32, 2001.

[10] Ali Geisa, Ronak Mehta, Hayden S. Helm, Jayanta Dey, Eric Eaton, Carey E. Priebe, and Joshua T. Vogelstein. Towards a theory of out-of-distribution learning. arXiv preprint at https://arxiv.org/abs/2109.14501, 2021.

[11] Albert Bifet and Ricard Gavalda. Adaptive learning from evolving data streams. In International Symposium on Intelligent Data Analysis, pages 249–260. Springer, 2009.

[12] Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. Mondrian forests: Efficient online

random forests. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems, volume 27. Curran Associates, Inc., 2014.

[13] Yael Ben-Haim and Elad Tom-Tov. A streaming parallel decision tree algorithm. Journal of Machine Learning Research, 11(28):849–872, 2010.

[14] Joshua T. Vogelstein, Jayanta Dey, Hayden S. Helm, Will LeVine, Ronak D. Mehta, Ali Geisa, Haoyin Xu, Gido M. van de Ven, Emily Chang, Chenyu Gao, Weiwei Yang, Bryan Tower, Jonathan Larson, Christopher M. White, and Carey E. Priebe. Omnidirectional transfer for quasilinear lifelong learning. arXiv preprint at https://arxiv.org/abs/2004.12908, 2021.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[16] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. R news, 2(3): 18–22, 2002.

[17] Gérard Biau, Luc Devroye, and Gäbor Lugosi. Consistency of random forests and other averaging classifiers. Journal of Machine Learning Research, 9(9), 2008.

[18] Martin Khannouz and Tristan Glatard. A benchmark of data stream classification for human activity recognition on connected objects. Sensors, 20(22), 2020.

[19] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. River: machine learning for streaming data in python, 2020.

[20] M Kumar. Scikit-garden/scikit-garden: A garden for scikit-learn compatible trees. Accessed: Apr, 13:2018, 2017.

[21] Molecular biology (splice-junction gene sequences). UCI Machine Learning Repository, 1991.

[22] E. Alpaydin and Fevzi. Alimoglu. Pen-based recognition of handwritten digits. UCI Machine Learning Repository, 1998.

[23] Alex Krizhevsky. Learning multiple layers of features from tiny images. University of Toronto, 05 2012.

[24] S. Rampone. Splice-junction recognition on gene sequences (dna) by brain learning algorithm. In 1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227), volume 1, pages 774–779 vol.1, 1998.

[25] Rahul Sarkar, Chandra Churh Chatterjee, Sayantan Das, and Dhiman Mondal. Splice junction prediction in dna sequence using multilayered rnn model. In Suresh Chandra Satapathy, K. Srujan Raju, K. Shyamala, D. Rama Krishna, and Margarita N. Favorskaya, editors, Advances in Decision Sciences, Image Processing, Security and Computer Vision, pages 39–47, Cham, 2020. Springer International Publishing.

[26] Allen Chieng Hoon Choong and Nung Kion Lee. Evaluation of convolutionary neural networks modeling of dna sequences using ordinal versus one-hot encoding method. In 2017 International Conference on Computer and Drone Applications (IConDA), pages 60–65, 2017.

[27] Fevzi Alimoglu and Ethem Alpaydin. Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition. In Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN 96, 1996.

[28] DM Roy and YW Teh. The mondrian process. In Advances in Neural Information Processing Systems, 2009.