On Imbalanced Regression with Hoeffding Trees

Pantia-Marina Alchirch[1](✉)[0000−0002−9596−7884] and Dimitrios I. Diochnos[1][0000−0002−2934−606X]

University of Oklahoma
{marina.alchirch,diochnos}@ou.edu

Abstract. Many real-world applications generate continuous data streams for regression. Hoeffding trees and their variants have a long-standing tradition due to their effectiveness, either alone or as base models in broader ensembles. Recent batch-learning work shows that *kernel density estimation (KDE)* improves smoothed predictions in imbalanced regression [46], while *hierarchical shrinkage (HS)* provides post-hoc regularization for decision trees without modifying their structure [1]. We extend KDE to streaming settings via a telescoping formulation and integrate HS into incremental decision trees. Empirical evaluation on standard online regression benchmarks shows that KDE consistently improves early-stream performance, whereas HS provides limited gains. Our implementation is publicly available at: https://github.com/marinaAlchirch/DSFA_2026.

Keywords: Online learning · Imbalanced regression · Hoeffding trees

1 Introduction

We live in a world where sensors continuously monitor the environment and our activities, generating data with the potential to transform our lives. In this context, machine learning and data mining algorithms are invaluable, as they uncover patterns that reveal important phenomena. Mining such continuous flows of information gives rise to *online* machine learning and *data stream* mining and has enabled applications such as credit scoring [3], fraud detection [29], human activity recognition [27], clinical decision support [18], energy pricing [42], and even reducing the energy footprint of stream mining itself [20].

At the same time, working with *imbalanced data* is a constant nuisance for machine learning and data mining applications. Imbalanced data arise when the labels associated with the observations that are collected, are skewed favoring certain categories or certain ranges of values. Research on imbalanced data has focused primarily on classification problems; e.g., anomaly detection [28], e-commerce [5], infants' survival [22], and many others. However, investigating imbalanced data makes a lot of sense in regression tasks as well. For example, one may want to predict the age of an individual by having access to their facial image [38], or predict weather phenomena such as hail size or wind intensity [32].

Our work focuses at the intersection of the above two fields. We investigate popular methods that generate decision trees incrementally and have been

developed with ideas for mining data streams and combine such mechanisms with recent advances developed for batch learning: *(i)* regularization of decision trees using hierarchical shrinkage [1], and *(ii)* approximating skewed distributions using kernel density estimation (KDE) [46]. In this work we are interested in imbalanced data; not in concept drift. Below we summarize our contributions.

Contributions. First, we implement hierarchical shrinkage on incremental decision trees that are available in the `scikit-multiflow` library [34]. Second, we revisit KDE and make it applicable to mining algorithms that operate on streams. Third, by combining the above techniques we achieve improved performance of incremental decision tree algorithms on imbalanced regression tasks for streams. Our main finding is that KDE helps with the improved predictions a lot, whereas hierarchical shrinkage provides minimal gains. We then integrate KDE with the `River` library [33] and verify that KDE improves predictions in the same tree models tested in `scikit-multiflow`, as well as in additional ones that were not tested or were not available in `scikit-multiflow`. Our code is publicly available for anyone to work with: https://github.com/marinaAlchirch/DSFA_2026.

2 Related Work

Incremental decision trees have a long line of research going back to at least the 80's [40,44]. Perhaps the most important incremental decision tree learning algorithm is that of Hoeffding trees [16] and their variants. Hoeffding trees use Hoeffding's inequality as a heuristic in order to decide when the difference in information gain (or some other relevant criterion) between the best-split attribute and the second-best split attribute is sufficiently high to justify a split. Several extensions of the Hoeffding trees have been developed [7,19,21,23,24,36], primarily to deal with concept drift, but attention has also been paid for parallelization [4], reduced complexity [2], or improved classification accuracy [31].

In parallel, a tremendous wealth of methods has been developed for mining imbalanced data. A well-known method for dealing with class imbalance is Synthetic Minority Oversampling Technique (SMOTE) in which oversampling of synthetic data is performed [11,43]. However, many more methods have been developed, including informed under-sampling the majority class [30], sampling based on clusters [25], re-weighting [45], modifying the margin of the separation boundary [9], or more theoretical approaches [10,13,15] including more general scenarios [12]. In addition, a recent line of work is using *kernel density estimation (KDE)* for smoothing the predicted labels on learned models [38,46], but has been applied only in batch settings where all the information is available ahead of time. Inspired by this approach we smooth the predicted labels using sketches of the distribution from small sequences of the stream examples.

Finally, a useful way for regularizing decision trees is that of *pruning* [8,39]. However, pruning is rarely used for incremental trees since it requires constructing a full tree and then pruning branches in a costly post-processing step. A more recent alternative for regularization is *hierarchical shrinkage (HS)* [1], where all

nodes along the root-to-leaf path contribute to the final prediction. HS is a post-hoc regularization method that avoids building a complete tree or performing expensive pruning operations, and can be applied to incremental trees provided sufficient streaming statistics are maintained. To our knowledge, we are the first to integrate HS into incremental decision trees and evaluate its impact on predictive accuracy, coupled with KDE-based smoothing for imbalanced domains.

3 Preliminaries

We use *multisets* for collections of objects that may contain multiple copies of the same object; e.g., $S = \{\{x_1, x_3, x_1\}\}$. For a compact representation denoting multiplicities, we may write $S = \{\{x_1, x_3, x_1\}\} = \{\{(x_1, 2), (x_3, 1)\}\}$. The *cardinality* $|S|$ of a multiset S is the total number of elements in S (including multiplicities); e.g., $|S| = 3$ for the multiset S given above. Using multisets we have the ability to approximate *probability density functions (pdfs)* that govern certain domains by keeping track of their *empirical densities*. Furthermore, in these cases we will explicitly indicate that certain elements do not appear at all, using 0 for their multiplicities. For example, in an instance space $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ using the collection of instances $S = \{\{(x_1, 2), (x_3, 1)\}\}$, we have the empirical density $\tilde{f}_S(\mathcal{X}) = \{(x_1, 2), (x_2, 0), (x_3, 1), (x_4, 0)\}$. With $m_{\mathcal{Z}}(z)$ we denote the multiplicity of z in $\mathcal{Z}$; e.g., $m_{\tilde{f}_S(\mathcal{X})}(x_1) = 2$ in the previous example.

Kernel Density Estimation (KDE). Kernels provide an easy way for similarity comparisons between instances. By storing instances in certain windows we can provide smooth estimates on imbalanced distributions for label values. In general, *kernel density estimation (KDE)* [37] using some arbitrary kernel K, points $z_i \in S$ from some multiset S, and a query point q, is accomplished via:

$$\hat{f}_{S,K,h}(q) = \frac{1}{|S|} \sum_{(z_i, m_i) \in S} m_i \cdot K_h(q - z_i) = \frac{1}{|S|\, h} \sum_{(z_i, m_i) \in S} m_i \cdot K\left(\frac{q - z_i}{h}\right), \quad (1)$$

where h is the *bandwidth* and its role is to provide some scaling on the use of the kernel that is used in (1). Small values of h make the estimate sensitive to individual data points, whereas large values of h provide a smoother estimate, by spreading the influence of each data point over a wider region of the dataset. We work with the Gaussian and the Epanechnikov kernel. The approximation on the labels as explained above is called LDS in [46].

Furthermore, we may use *binning* to map individual label values to broader intervals. This is standard technique; e.g., [46]. Let m and M denote the minimum and maximum label values (assumed known), and let r be the bin range. We generate $j^\star = \lfloor (M - m)/r \rfloor + 1$ bins, where $b_j = [m + jr, m + (j + 1)r)$ for $j \in \{0, 1, \ldots, j^\star\}$. Since (1) ultimately computes an average, it admits a telescoping update that relies on the previous average and the newest observation; which is particularly convenient in streaming settings. Using n in the first subscript to

Algorithm 1 Incremental KDE

Require: Window $W = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, bandwidth h, kernel K, and a structure of B groups of bins.

Ensure: Smoothed weights $\hat{f}_{n,K,h}(b)$ for each bin $b \in B$, where n is number of examples that we have seen in the stream so far.

1: **for** target label $y_i \in W$ **do**
2: Find bin $b = \lfloor (y_i - m)/r \rfloor$ for target label y_i
3: Compute $\hat{f}_{n+i,K,h}(b)$ as shown in (2)
4: **return** $\{\hat{f}_{n+m,K,h}(b)\}$, for all b $\in B$

denote the fact that we have seen n examples so far from the stream, we have:

$$\hat{f}_{n,K,h}(q) = \hat{f}_{n-1,h}(q) + \frac{1}{n}\left(\frac{1}{h}K\left(\frac{q - z_n}{h}\right) - \hat{f}_{n-1,h}(q)\right). \tag{2}$$

Thus, we are able to implement KDE incrementally; Algorithm 1 has the details.

Hoeffding Trees and Variants. The most well-known incremental decision tree algorithm is the Hoeffding Tree (HT) [16] and its variants. This class of algorithms uses Hoeffding's inequality to decide when to split a node. Letting R be the range of the split evaluation function (e.g., $R = 1$ for information gain in binary classification), when the difference between the best and second-best attributes exceeds $\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$, then, with confidence at least $1 - \delta$, the best attribute is selected for splitting. For our work we use the `scikit-multiflow` [34] and `River` [33] libraries. The `scikit-multiflow` library has an implementation of HT and of the *Hoeffding Adaptive Tree (HAT)* [7]. HAT employs ADWIN [6] to detect potential concept drift. Similar to `scikit-multiflow` the `River` library has an implementation of HT and HAT. Sticking to regression, `River` also has an implementation of iSOUP [36] and SGT [21].

Regularization of Decision Trees Using Hierarchical Shrinkage. Hierarchical shrinkage (HS) [1] is a recent regularization technique that does not alter the structure of the learned tree. Letting $N(t)$ be the number of samples in a node t of a decision tree DT and $\mathbf{E}_t[y]$ to be the mean response in node t, the prediction on an instance x is $DT(x) = \mathbf{E}_{t_0}[y] + \sum_{l=1}^{L}\left(\mathbf{E}_{t_l}[y] - \mathbf{E}_{t_{l-1}}[y]\right)$. HS modifies this to

$$DT_\lambda(x) = \mathbf{E}_{t_0}[y] + \sum_{l=1}^{L} \frac{\mathbf{E}_{t_l}[y] - \mathbf{E}_{t_{l-1}}[y]}{1 + \lambda/N(t_{l-1})}, \tag{3}$$

for some hyper-parameter λ that performs the regularization. Thus, the nodes in the path from root to leaf to have an amount of say in the final prediction.

4 The Online Learning Process That We Study

We use a variation of the *Follow-the-Leader (FTL)* algorithm [41] for tuning. In FTL, multiple models run in parallel and, at each time step, prediction follows

Algorithm 2 Online-Learning Process & Follow-the-Leader Tuning

Require: Stream $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n), \ldots\}$ of labeled examples, a set P of parameters, a tuning window size s_t, and no-tuning window size s'_t.

```
1:  Initialize a model M;
2:  M.parameters ← ∅;
3:  Initialize all different models T based on P;
4:  i ← 1;
5:  tuneStart ← 1;
6:  tuneEnd ← s_t;
7:  for every example (x_i, y_i) in stream S do
8:      if i ≥ tuneStart and i ≤ tuneEnd then
9:          j ← 1;
10:         minError ← ∞;
11:         bestModel ← ∅;
12:         while j ≤ s_t do
13:             Predict-then-train with every model t ∈ T;
14:             if minError ≥ t.error then
15:                 minError ← t.error;
16:                 bestModel ← t;
17:             j ← j + 1;
18:         i ← i + s_t;
19:         tuneStart ← tuneStart + s_t + s'_t;
20:         tuneEnd ← tuneStart + s_t − 1;
21:         M.setParameters(bestModel.parameters);
22:     else
23:         Predict-then-Train on example (x_i, y_i) with M;
24:         i ← i + 1;
```

the model that performs best (e.g., lowest cumulative loss) at that point in time. To ease the complexity and memory requirements we periodically allocate portions of the stream for such tuning and, after each phase, we select the hyper-parameters that performed best in the most recent tuning period. This procedure is summarized in Algorithm 2 and explained further below.

Tuning and learning on the stream is governed by two parameters: a parameter s_t controlling the duration of each tuning phase, and a parameter s'_t specifying how many examples are processed in a predict-then-train manner between tuning phases. The model used for prediction is denoted by M; it is selected as the best-performing model from a pool of models T. In Line 3, we initialize T with all instantiations of the base learner (e.g., Hoeffding Tree) across all combinations of hyper-parameter values. Hence, during tuning we effectively perform a grid search, where each grid point corresponds to the same base model with different hyper-parameters. Unlike standard grid search, all models in T are trained in parallel on the stream. For example, one configuration may consist of a Hoeffding Tree combined with KDE using a Gaussian kernel with bandwidth h, bin range r, and no hierarchical shrinkage ($\lambda = 0$).

Table 1: Datasets and potential preprocessing for the experiments. Online tuning was tried in all datasets except for Semi due to time constraints.

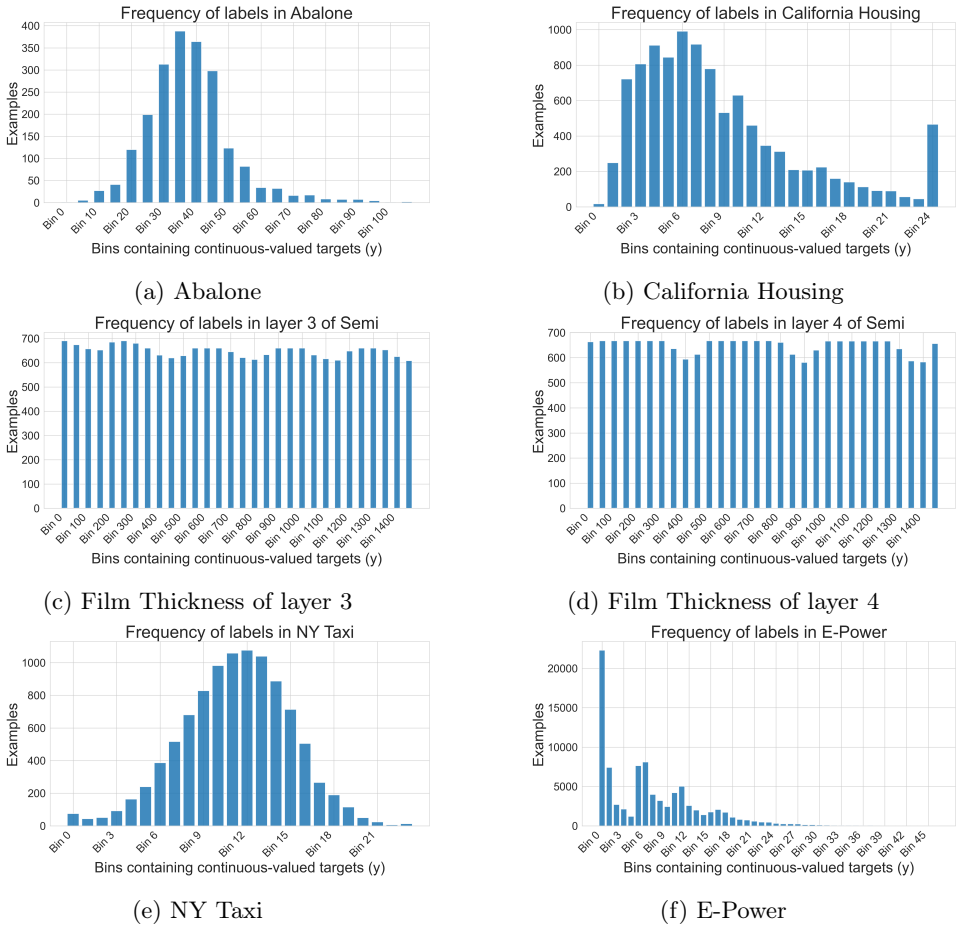
Dataset	#Features	#Examples	Predict-then-train set size used in Algorithm 2	Outlier removal	Constructed feature
Abalone	8	4177	2089	✗	✗
Semi	229	19452	-	✓	✗
California	8	20635	10319	✓	✗
NY Taxi	10	19997	10001	✓	✓
E-Power	8	99992	87992	✓	✓

5 Experiments

Datasets. We conduct experiments on Abalone [17], California Housing (California) [26], Electric Power Consumption (E-Power) [17], New York Taxi (NY Taxi) [35], and Film Thickness in semiconductors (Semi) [14]; see Table 1 for dataset characteristics. Abalone is small and we are using it "as-is". For the remaining datasets we remove outliers, and for NY Taxi and E-Power we additionally construct standard features (e.g., Manhattan distance for NY Taxi; hour and weekday for E-Power). For California Housing, we use 20,635 examples after removing 5 extreme instances. The NY Taxi dataset (currently has 1,458,644 examples) is treated as a stream using the first 20,000 instances, with 3 outliers removed. Similarly, from E-Power (2,075,259 examples) we use the first 100,000 instances and remove 8 outliers. Finally, from the Semi dataset (810,000 examples) we use the first 20,000 instances, retaining 19,452 after outlier removal. Figure 1 illustrates target imbalance in all datasets using bin range $r = 0.2$.

Clarification on the Integration with Python Libraries and Datasets Tested. In our initial submission we integrated HS and KDE with the `scikit-multiflow` library. When we started our work, `scikit-multiflow` exposed more of the internal structure of its learning algorithms, especially Hoeffding Trees, making it easier to integrate HS directly into the learning procedure. For example, in HAT, HS only requires the root-to-leaf prediction path, since the statistics of intermediate nodes are directly accessible regardless of additional subtrees used for drift detection. Meanwhile, the `River` library was further developed and has a more current version of iSOUP, as well as SGT which does not exist in `scikit-multiflow`. After receiving the reviews, we integrated KDE with `River` and ran additional experiments; however, without hyperparameter tuning due to time constraints.

Furthermore, in our initial submission we ran experiments only against three datasets: California, NY Taxi, and E-Power. For the camera-ready version of the paper we decided to run experiments with Abalone too, as well as against the more challenging layers 3 and 4 of the Semi dataset. Abalone being fairly



(a) Abalone (b) California Housing

(c) Film Thickness of layer 3 (d) Film Thickness of layer 4

(e) NY Taxi (f) E-Power

Fig. 1: Frequency of labels in predict-then-train data using bins' range $r = 0.2$.

small in size allowed us to run on time Algorithm 2 on scikit-multiflow, but due to time constraints we could not do the same thing with Semi. Experiments on layers 1 and 2 of Semi were omitted, as these cases appear trivial or easy and the KDE variants consistently outperform the baseline models.

Thus, we have experiments with `scikit-multiflow` -testing both HS and KDE- using the tuning phases of Algorithm 2 in all datasets but on Semi, while we have experiments with the `River` library in all datasets -testing for potential benefit of KDE only- but without tuning this time.

Using `scikit-multiflow` with Algorithm 2 we use HT and HAT as base models. We equip both of these models with KDE for smoothing the imbalanced label distribution of the stream as well as HS for regularizing the predictions. These enhancements appear either together or in isolation. Regarding incremental KDE we use a tumbling window whose size we treat as a hyper-parameter.

On the Tuning Window and Hyper-Parameter Tuning. When running experiments with the `scikit-multiflow` integration we use four periods for tuning the hyper-parameters of our models. For a dataset S that we treat as a stream, we set the tuning window size to be $\min\left(\lfloor |S|/8 \rfloor, 3000\right)$. Based on Table 1, the tuning window size for each dataset tested is approximately one quarter of the "missing" examples from the table. For example, in the case of Abalone, this translates to $(4,177 - 2089)/4 = 522$ examples for each tuning phase. In each tuning window we choose from the following hyperparameter values: *(i)* for range of bin $r \in \{0, 0.1, 0.2, 0.5, 1\}$, where 0 means that we don't perform binning in our targets but every target is simply its own bin, *(ii)* $\lambda \in \{0, 0.1, 1, 10, 15, 25\}$, where 0 is for not predicting with HS but rather with the initial unregularized model, *(iii)* $h \in \{10, 50, 100\}$, *(iv)* for (tumbling) window size $|W| \in \{50, 100, 200\}$, and *(v)* for kernel choose between Gaussian and Epanechnikov. When we use KDE alone we do not tune on λ, while when we use HS alone we only tune on λ.

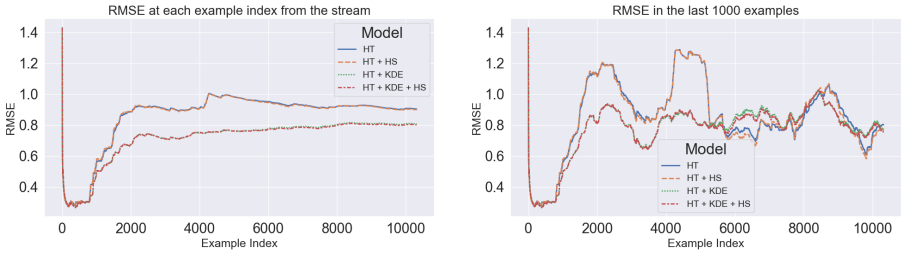
Evaluation Metrics. For evaluating our experiments we use: *mean absolute error (MAE)*, *root mean squared error (RMSE)*, *weighted root mean squared error (WRMSE)*, and the R^2 statistic. For the WRMSE the penalty calculated on a set of test points S is $\mathrm{WRMSE}(f, S) = \sqrt{\sum_{i=1}^{|S|} w(y)(f(x) - y)^2}$, where $w(y)$ is the weight associated with the true label y that corresponds to the input x. In particular, this weight $w(y)$ corresponds to the probability of the bin where y falls, as shown in Figure 1 (after normalization so that we have a probability distribution). Notice that the number of bins is in principle much smaller than the number of examples $|S|$ and hence the weight $w(y)$ will be much larger than $1/|S|$ for the non-empty bins that contribute in the WRMSE value. For example, if we have bins with weight values 0.1 or larger, then the WRMSE will be several orders of magnitude larger than simple RMSE for a datastream that has processed $n = 10,000$ examples and therefore every example in this stream contributes to RMSE a value with weight equal to $w(y) = 1/10,000$. In the results that follow we will see this phenomenon. For the selection of the best hyper-parameters during tuning, we use the (non-weighted) RMSE.

Experimental Results on `scikit-multiflow`. Table 2 presents the overall results obtained using the `scikit-multiflow` library. For this reason, we report results only for HT and HAT, along with their HS, KDE, and KDE+HS extensions. Our results in Table 2 indicate that in all datasets except in E-Power our KDE variants (potentially with HS as well) outperform their original versions.

We now provide some additional information complementing the results of Table 2. In Figure 2 we see that for the California dataset, the performance of the KDE and the KDE+HS variants of HT outperform the original model, with the largest performance gap appearing in the middle of the stream. In Figure 3 we see the MAE performance of HAT models on the Abalone dataset. This time the KDE+HS enhanced version of HAT behaves the best and is discernibly better compared to the simple KDE extension of the HAT model. In Figure 4 we see two graphs of R^2 for the NY Taxi and the E-Power datasets using again HAT-based models for comparisons. In both cases we see that the KDE-enhanced and

Table 2: Performance comparison across datasets on the `scikit-multiflow` library using Algorithm 2 for tuning. **Bold blue** indicates best metric value per dataset, whereas plain **bold** indicates second best. For comparison purposes on plain HT and HAT we bypass the tuning portions of the stream.

		Model (potentially including HS, KDE, or both)							
Dataset	**Metric**	**HT**	**HAT**	**HT +HS**	**HAT +HS**	**HT +KDE**	**HAT +KDE**	**HT +HS +KDE**	**HAT +HS +KDE**
Abalone	MAE	1.979	1.979	1.979	1.979	**1.643**	1.648	1.644	**1.560**
	RMSE	2.634	2.634	2.634	2.634	2.350	2.350	**2.347**	**2.233**
	WRMSE	**7.037**	**7.037**	**6.916**	**6.916**	7.255	7.508	7.210	7.471
	R^2	-0.015	-0.015	-0.015	-0.015	0.192	0.192	**0.194**	**0.271**
California	MAE	0.668	0.692	0.659	0.700	**0.568**	0.625	**0.564**	0.628
	RMSE	0.904	0.931	0.899	0.940	**0.808**	0.914	**0.802**	0.915
	WRMSE	20.282	21.793	19.755	21.966	**18.851**	22.148	**18.557**	22.227
	R^2	0.380	0.342	0.387	0.330	**0.505**	0.367	**0.512**	0.366
NY Taxi	MAE	0.466	0.466	0.466	0.466	**0.368**	**0.368**	**0.369**	**0.369**
	RMSE	0.612	0.612	0.612	0.612	**0.487**	**0.487**	**0.487**	**0.487**
	WRMSE	13.177	13.177	13.147	13.147	**11.932**	**11.932**	**11.786**	**11.786**
	R^2	0.333	0.333	0.333	0.333	**0.577**	**0.577**	**0.577**	**0.577**
E-Power	MAE	**0.080**	0.407	**0.098**	0.419	0.114	0.229	0.114	0.230
	RMSE	**0.203**	0.660	**0.219**	0.666	0.272	0.479	0.272	0.479
	WRMSE	**9.460**	41.994	**12.371**	43.223	15.299	29.430	15.310	29.479
	R^2	**0.977**	0.756	**0.973**	0.752	0.959	0.872	0.959	0.872

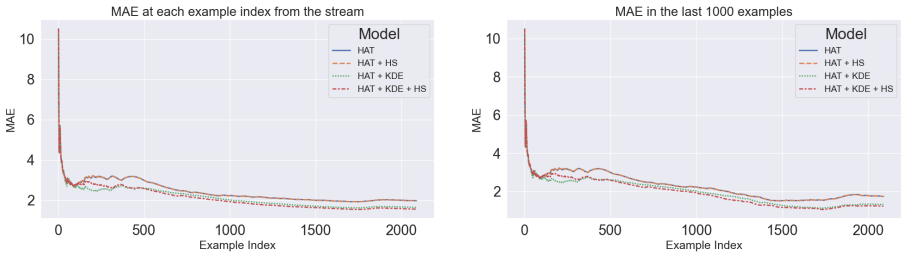


(a) Cumulative RMSE.

(b) RMSE in the last 1K examples.

Fig. 2: RMSE plots of HT-based models for the *California* dataset. The KDE and KDE+HS variants perform overall better than the original HT model.

KDE+HS-enhanced variants perform better than the base model. Having said that, for the case of E-Power, the simple HT performs even better (see Table 2).



(a) Cumulative MAE. (b) MAE in the last 1K examples.

Fig. 3: MAE plots of HAT-based models for the *Abalone* dataset. Similar to Figure 2 the KDE-enhanced model performs better than the original model. However, here HS helps even further.

(a) R^2 in the last 1K examples (NY Taxi). (b) R^2 in the last 1K examples (E-Power).

Fig. 4: R^2 score during the last 1K examples for HAT-based models on the NY Taxi (Fig. 4a) and E-Power (Fig. 4b) datasets. In both cases, the KDE variants achieve the best performance throughout the streaming process. Furthermore, in both cases plotting starts after 100 examples have first been processed since the early R^2 values are very small and dominate the y-axis in the graph.

Experimental Results on `River`. Following the reviewers' feedback, we executed additional experiments using the `River` Python library as well. This allowed us to evaluate our KDE framework with the iSOUP and SGT models beyond HT and HAT that we evaluated using `scikit-multiflow`. Due to time constraints, we did not perform online tuning using Algorithm 2. Instead, we only performed incremental KDE as described in Algorithm 1, using the same fixed parameters ($h = 10$, $r = 0.2$ and window size $|W| = 50$) for all the experiments. Table 3 presents results with the `River` library, using both the RMSE and the *weighted* RMSE (WRMSE) metrics. We denote layers 3 and 4 of the Semi dataset as Semi-L3 and Semi-L4 respectively. As we had no plots regarding the Semi dataset generated using models from `scikit-multiflow`, in Figure 5 we now plot two graphs of the Semi dataset focusing on layers 3 and 4. Regarding performance in layer 3 in Figure 5a HT performs better using KDE, but in Figure 5b the KDE

Table 3: Performance metric comparison (lower is better) across datasets using the `River` Python library. **Bold** values indicate the better performance between each base model and its KDE-enhanced counterpart. **Bold blue** indicates the best performing model for a given dataset and metric. No tuning has been performed to identify the best hyper-parameters. For the results shown for Semi-L3 and Semi-L4 we are using only one decimal digit for better clarity.
In the case of RMSE KDE helps in 18 out of 24 cases when comparing a base model and its KDE-enhanced version. Also, when using RMSE in five out of six datasets the overall best performing model is some base model enhanced with KDE (shown in **bold blue** per row).
In the case of weighted RMSE (WRMSE) KDE helps in 15 out of 24 cases when comparing a base model and its KDE-enhanced version. Also, when using WRMSE in four out of six datasets the overall best performing model is some base model enhanced with KDE (shown in **bold blue** per row).

		Model (potentially including KDE)							
Dataset	**Metric**	**HT**	**HAT**	**iSOUP**	**SGT**	**HT +KDE**	**HAT +KDE**	**iSOUP +KDE**	**SGT +KDE**
Abalone	RMSE	3.175	3.186	**11.249**	4.051	**2.776**	**2.633**	11.318	**3.138**
	WRMSE	**43.815**	**42.957**	**215.832**	64.202	50.971	46.795	218.941	**57.864**
California	RMSE	0.837	0.827	2.367	1.033	**0.768**	**0.770**	**2.366**	**0.940**
	WRMSE	26.069	26.725	70.863	32.971	**24.118**	**24.753**	**70.852**	**29.310**
NY Taxi	RMSE	0.553	0.525	6.494	0.829	**0.485**	**0.491**	**6.489**	**0.511**
	WRMSE	17.494	**16.797**	254.611	30.739	**16.664**	16.981	**254.379**	**17.625**
E-Power	RMSE	0.160	0.672	2.124	**0.477**	**0.082**	**0.403**	**2.124**	0.574
	WRMSE	11.136	50.203	112.694	**38.406**	**4.074**	**25.534**	**112.657**	52.591
Semi-L3	RMSE	113.1	80.3	**174.8**	**85.2**	**82.8**	**55.4**	175.9	91.3
	WRMSE	2866.6	2058.1	**4421.1**	**2174.6**	**2114.7**	**1432.0**	4449.2	2339.2
Semi-L4	RMSE	**84.7**	**85.4**	176.2	88.7	88.9	95.4	**176.0**	**88.2**
	WRMSE	**2158.4**	**2177.3**	4464.8	2261.4	2265.6	2430.4	**4460.9**	**2250.9**

variant is underperforming during most of the streaming process and becomes equivalent to plain HT near the end.

Coming back to Table 3, a first remark is that KDE is in general helpful for the base models. In the case of RMSE, the KDE-enhanced models have better performance in 18 out of 24 cases, while in the case of WRMSE the KDE-enhanced models have better performance in 15 out of 24 cases. Overall, depending on the performance metric, RMSE or weighted RMSE, the KDE-enhanced models provide the best performing models in five out of six datasets and four out of six datasets respectively.

Resources Used for Experiments. All experiments were executed on a macOS laptop with an Apple Silicon M2 processor and 16 GB of RAM. We implemented the models in Python using the `scikit-multiflow` and `River` libraries.

(a) Cumulative RMSE in Semi-L3. (b) RMSE for 1K last examples in Semi-L4.

Fig. 5: RMSE, cumulative and during the last 1K examples, for HT-based models on the Semi-L3 (Fig. 5a) and Semi-L4 (Fig. 5b) datasets. Note that these results were obtained using the `River` library.

6 Conclusions and Future Work

We recast recent advances in *batch learning* regarding imbalanced data and regularization of decision trees, in *streaming scenarios*. We propose the inclusion of KDE and potentially HS in incremental decision tree learning algorithms. We have seen that, in general, KDE improves the performance of the base models. However, HS appears to be providing minimal gains, if at all. As our results improve the performance of base incremental decision trees used for regression, we anticipate that the observed benefits will translate to similar benefits in random forests, or other ensembles of such tree-based models. We note that KDE does not naturally extend to pure classification problems and it is a question if one can find a new method that can further improve the performance of such incremental models with, or without, the inclusion of HS. Furthermore, in this work we have not focused on drift, which is an exciting direction, especially when one would like to couple drift together with imbalanced data and potentially KDE.

Acknowledgments. This material is based upon work supported by the U.S. National Science Foundation under Grant No. RISE-2019758. This work is part of the NSF AI Institute for Research on Trustworthy AI in Weather, Climate, and Coastal Oceanography (NSF AI2ES).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Agarwal, A., Tan, Y.S., Ronen, O., Singh, C., Yu, B.: Hierarchical Shrinkage: Improving the accuracy and interpretability of tree-based models. In: ICML. Proceedings of Machine Learning Research, vol. 162, pp. 111–135. PMLR (2022)
2. Barddal, J.P., Enembreck, F.: Learning regularized hoeffding trees from data streams. In: SAC. pp. 574–581. ACM (2019)

3. Barddal, J.P., Loezer, L., Enembreck, F., Lanzuolo, R.: Lessons learned from data stream classification applied to credit scoring. Expert Syst. Appl. **162**, 113899 (2020)
4. Ben-Haim, Y., Tom-Tov, E.: A Streaming Parallel Decision Tree Algorithm. J. Mach. Learn. Res. **11**, 849–872 (2010)
5. Besaleva, L., Weaver, A.C.: Classification of imbalanced data in E-commerce. In: 2017 IntelliSys. pp. 744–750. IEEE (2017)
6. Bifet, A.: Adaptive learning and mining for data streams and frequent patterns. SIGKDD Explor. Newsl. **11**(1), 55–56 (Nov 2009)
7. Bifet, A., Gavaldà, R.: Adaptive Learning from Evolving Data Streams. In: Adams, N.M., Robardet, C., Siebes, A., Boulicaut, J. (eds.) IDA 2009. LNCS, vol. 5772, pp. 249–260. Springer (2009)
8. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth (1984)
9. Cao, K., Wei, C., Gaidon, A., Aréchiga, N., Ma, T.: Learning Imbalanced Datasets with Label-Distribution-Aware Margin Loss. In: NeurIPS. pp. 1565–1576 (2019)
10. Chaudhuri, K., Ahuja, K., Arjovsky, M., Lopez-Paz, D.: Why does Throwing Away Data Improve Worst-Group Error? In: ICML. Proceedings of Machine Learning Research, vol. 202, pp. 4144–4188. PMLR (2023)
11. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research **16**, 321–357 (2002)
12. Cohen, L., Mansour, Y., Moran, S., Shao, H.: Probably Approximately Precision and Recall Learning. CoRR **abs/2411.13029** (2024)
13. Cortes, C., Mao, A., Mohri, M., Zhong, Y.: Balancing the Scales: A Theoretical and Algorithmic Framework for Learning from Imbalanced Data. In: ICML. Proceedings of Machine Learning Research, vol. 267. PMLR / OpenReview.net (2025)
14. DACON Co., Ltd.: Monthly Semiconductor Thin Film Thickness Analysis Dataset. https://dacon.io/competitions/official/235554/data (2020)
15. Diochnos, D.I., Trafalis, T.B.: Learning Reliable Rules under Class Imbalance. In: SDM. pp. 28–36. SIAM (2021)
16. Domingos, P.M., Hulten, G.: Mining high-speed data streams. In: Ramakrishnan, R., Stolfo, S.J., Bayardo, R.J., Parsa, I. (eds.) KDD. pp. 71–80. ACM (2000)
17. Dua, D., Graff, C.: UCI machine learning repository (2017)
18. Fong, S., Zhang, Y., Fiaidhi, J., Mohammed, O., Mohammed, S.: Evaluation of stream mining classifiers for real-time clinical decision support system: a case study of blood glucose prediction in diabetes therapy. BioMed Res. Intl. **1**, 274193 (2013)
19. Gama, J., Fernandes, R., Rocha, R.: Decision trees for mining data streams. Intell. Data Anal. **10**(1), 23–45 (2006)
20. García-Martín, E., Bifet, A., Lavesson, N.: Energy modeling of Hoeffding tree ensembles. Intell. Data Anal. **25**(1), 81–104 (2021)
21. Gouk, H., Pfahringer, B., Frank, E.: Stochastic Gradient Trees. In: ACML. Proceedings of Machine Learning Research, vol. 101, pp. 1094–1109. PMLR (2019)
22. Gourdeau, P., Kanbar, L.J., Shalish, W., Sant'Anna, G.M., Kearney, R.E., Precup, D.: Feature selection and oversampling in analysis of clinical data for extubation readiness in extreme preterm infants. In: EMBC. pp. 4427–4430 (2015)
23. Hulten, G., Spencer, L., Domingos, P.M.: Mining Time-Changing Data Streams. In: KDD. pp. 97–106. ACM (2001)
24. Ikonomovska, E., Gama, J., Dzeroski, S.: Learning model trees from evolving data streams. Data Min. Knowl. Discov. **23**(1), 128–168 (2011)

25. Jo, T., Japkowicz, N.: Class imbalances versus small disjuncts. SIGKDD Explorations **6**(1), 40–49 (2004)
26. Kelley Pace, R., Barry, R.: Sparse spatial autoregressions. Statistics & Probability Letters **33**(3), 291–297 (1997)
27. Khannouz, M., Glatard, T.: A Benchmark of Data Stream Classification for Human Activity Recognition on Connected Objects. Sensors **20**(22), 6486 (2020)
28. Khreich, W., Granger, E., Miri, A., Sabourin, R.: Iterative Boolean combination of classifiers in the ROC space: An application to anomaly detection with HMMs. Pattern Recognition **43**(8), 2732–2752 (2010)
29. Laleh, N., Azgomi, M.A.: A Taxonomy of Frauds and Fraud Detection Techniques. In: ICISTM. CCIS, vol. 31, pp. 256–267. Springer (2009)
30. Liu, X., Wu, J., Zhou, Z.: Exploratory Undersampling for Class-Imbalance Learning. IEEE Transactions on Systems, Man, and Cybernetics Part B **39**(2), 539–550 (2009)
31. Manapragada, C., Webb, G.I., Salehi, M.: Extremely Fast Decision Tree. In: KDD. pp. 1953–1962. ACM (2018)
32. McGovern, A., Elmore, K.L., Gagne, D.J., Haupt, S.E., Karstens, C.D., Lagerquist, R., Smith, T., Williams, J.K.: Using artificial intelligence to improve real-time decision-making for high-impact weather. Bulletin of the American Meteorological Society **98**(10), 2073 – 2090 (01 Oct 2017)
33. Montiel, J., Halford, M., Mastelini, S.M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H.M., Read, J., Abdessalem, T., Bifet, A.: River: machine learning for streaming data in Python. J. Mach. Learn. Res. **22**, 110:1–110:8 (2021)
34. Montiel, J., Read, J., Bifet, A., Abdessalem, T.: Scikit-Multiflow: A Multi-output Streaming Framework. J. Mach. Learn. Res. **19**, 72:1–72:5 (2018)
35. New York City Taxi and Limousine Commission: Tlc trip record data. https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page (2009-2025)
36. Osojnik, A., Panov, P., Dzeroski, S.: Tree-based methods for online multi-target regression. J. Intell. Inf. Syst. **50**(2), 315–339 (2018)
37. Parzen, E.: On estimation of a probability density function and mode. The Annals of Mathematical Statistics **33**(3), pp. 1065–1076 (1962)
38. Pu, R., Xu, G., Fang, R., Bao, B., Ling, C., Wang, B.: Leveraging Group Classification with Descending Soft Labeling for Deep Imbalanced Regression. In: AAAI. pp. 19978–19985. AAAI Press (2025)
39. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
40. Schlimmer, J.C., Fisher, D.H.: A Case Study of Incremental Concept Induction. In: AAAI. pp. 496–501. Morgan Kaufmann (1986)
41. Shalev-Shwartz, S.: Online learning and online convex optimization. Found. Trends Mach. Learn. **4**(2), 107–194 (Feb 2012)
42. Sun, Y., Gomes, H.M., Pfahringer, B., Bifet, A.: Real-Time Energy Pricing in New Zealand: An Evolving Stream Analysis. In: PRICAI. LNCS, vol. 15285, pp. 91–97. Springer (2024)
43. Torgo, L., Ribeiro, R.P., Pfahringer, B., Branco, P.: Smote for regression. In: Portuguese conference on artificial intelligence. pp. 378–389. Springer (2013)
44. Utgoff, P.E.: ID5: An Incremental ID3. In: ICML. pp. 107–120 (1988)
45. Wang, Y., Ramanan, D., Hebert, M.: Learning to Model the Tail. In: NeurIPS. pp. 7029–7039 (2017)
46. Yang, Y., Zha, K., Chen, Y.C., Wang, H., Katabi, D.: Delving into Deep Imbalanced Regression. In: ICML. PMLR, vol. 139, pp. 11842–11851. PMLR (2021)