Active Learning Embedded in Incremental Decision Trees

Vinicius Eiji Martins[1(✉)], Victor G. Turrisi da Costa[2], and Sylvio Barbon Junior[1]

[1] Computer Science Department, Londrina State University, Londrina, PR, Brazil
{vinicius.martins,barbon}@uel.br
[2] DISI, University of Trento, Trento, Italy
vg.turrisidacosta@unitn.it

Abstract. As technology evolves and electronic devices become widespread, the amount of data produced in the form of stream increases in enormous proportions. Data streams are an online source of data, meaning that it keeps producing data continuously. This creates the need for fast and reliable methods to analyse and extract information from these sources. Stream mining algorithms exist for this purpose, but the use of supervised machine learning is extremely limited in the stream domain since it is unfeasible to label every data instance requested to be processed. Tackling this problem, our paper proposes the use of active learning techniques for stream mining algorithms, specifically incremental Hoeffding trees-based. It is important to mention that the active learning techniques were implemented to match the stream mining constraints regarding low computational cost. We took advantage of the incremental tree original structure to avoid overburdening the original computational cost when selecting a label. In other words, the statistical strategy to grow each incremental tree has supported the execution of active learning. Using techniques of uncertainty sampling, we were able to drastically reduce the number of labels required at the cost of a very small reduction in accuracy. Particularly with *Budget Entropy* there was an average negative impact of accuracy about 4% using only 14% of samples labelled.

Keywords: Stream mining · Active learning · Hoeffding trees

1 Introduction

Data streams are an increasingly common resource that produces a large amount of potentially infinite data in short intervals. Dealing with this type of data,

The authors would like to thank the financial support of the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, the National Council for Scientific and Technological Development (CNPq) of Brazil - Grant of Project 420562/2018-4 - and Fundação Araucária.

stream mining algorithms need to face a set of challenges, such as where and how to store data, how to process data in an acceptable time frame and how to deal with its changes in concepts and underlying distributions [11,22].

A common way to extract useful information and patterns from data is through the use of supervised machine learning models, including the decision trees. Incremental decision trees are alternatives from supervised machine learning algorithms for data stream scenarios, in which each single stream sample can be used to update a decision tree. A classic example of incremental decision trees is the Hoeffding Tree (HT) [11], which is based on the Hoeffding Bounds (HB) theory to identify the best split feature during tree growth. HB has gained notoriety in the stream mining scenario for its effectiveness, a fact that impelled its usage in several implementations of incremental decision trees such as Very Fast Decision Tree (VFDT) [11] and Strict Very Fast Decision Tree [6] (SVFDT), the latter focuses on reducing the requirement of computational resources.

However, all these algorithms rely on labelled data, which may be expensive to acquire in the real world and even harder to gather in data stream situations. The challenges of volume and velocity intensify the problem of labelling samples from data streams. Some techniques were developed to overcome this problem for traditional supervised machine learning, such as Semi-Supervised Learning (SSL) [32] and Active Learning (AL) [14]. SSL assumes a small amount of labelled data and a large pool of unlabelled data and uses both to train its model. On the other hand, AL works by intelligently selecting only a subset of data samples to be labelled, allowing the algorithms to train efficiently in more realistic conditions [26].

AL does not require labelled instances before the training begins as it will choose which data instances it will learn from. This is done by asking *queries* containing unlabelled data to an *oracle* that informs it the true label, in several cases the oracle could be a human specialist [26]. With the application of this technique, the model only needs to be trained on a small number of highly informative samples instead of the whole dataset, increasing the efficiency of the training with minimal accuracy losses, following the constraints of reduced access to the complete dataset. AL techniques are broken into various categories, but all are grounded in the same idea: using a sampling technique, the most informative instances are selected or constructed from the input domain and sent to an oracle, human or machine, that will label it and return to the learner, which will then use it to train in a supervised manner.

In recent years, several efforts have been made in the direction to create joint methods with data stream and AL [1,10,19–21,28]. The goal of the major part of the proposed algorithms is to tackle Concept Drift and the detection of Novelty. Concept drift reflects the idea that concepts in the real world are always changing.

Concept drift was addressed using ensemble learning by some works [1,20,28]. Ensemble learning is an important solution used in the stream mining community since they maintain the advantages present in traditional scenarios, such as taking advantage of local competencies from classifiers and robustness to overfitting [7]. Also, ensembles can handle the drifting context ensured by the diversity of committee members.

In [1] the authors propose a framework for use of AL in ensembles with their proposed Query-by-bagging and Query-by-boosting methods, both based upon the paradigm of Query by Committee(QBC) [27]. These methods make the oracle responsible for choosing the data samples which will be labelled and appended to the training dataset that will be broken into various windows that will be used by the ensemble learners to train its models. Besides the fact that concept drifts are not addressed, the framework has a cost of an additional structure.

Shan et al. [28] proposes a framework for ensemble active learning using an ensemble composed of a stable permanent classifier that learns from every data instance that arrives and multiple dynamic ephemeral classifiers that only train with a limited amount of data. A combination of Uncertainty Sampling and Random Sampling is used to determine if a sample inside the data block will be labelled and used to train the ensemble. This combination of a permanent classifier and multiple short lived classifiers make this framework able to adapt to sudden and gradual concept drifts while reducing labeling costs by focusing the queries to the oracle when drift occurs.

In [20] an approach to ensemble active learning is proposed that instead of selecting instances to query based on the amount of disagreement between committee members, it uses a Multi-armed bandit approach, where the most competent member is made responsible for this decision. This approach allows the ensemble to better adapt under concept drifts, specifically when drifts occur in regions of data that regular query sampling techniques register low amounts of uncertainty.

These approaches [1,20,28] present novel ensemble techniques adapted to AL and streaming situations, but they introduce additional complexity and costs to the training procedure.

Alternatively to ensembles, [10] proposed a sequential ID3, grounded on a sequential probability ratio evaluation to reduce the number of samples sufficient to perform a split. They affirm that no theoretical bounds are exposing the extent to which labels can be saved without significantly compromising performance. However, the AL strategy used in [10] has a cost of memory and additional mechanism of control the selection of samples to be labelled. Furthermore, the authors described the implementation of AL with VFDT as a promising approach.

In this work we combine the use of three Hoeffding Tree implementations with AL strategies. The implementations are all variations of the VFDT and SVFDT (SVFDT-I and SVFDT-II). They are robust learners that deal very well with various streaming situations while also performing memory management [6], but they are still supervised machine learning techniques that expect labelled data in the stream. We compared two strategies of AL literature, *Entropy-based* [29] and *Budget Entropy* [34], and proposed a novel mechanism called *Best Budget Entropy*. We took advantage of original implementation from the most memory-friendly HT algorithms to avoid consuming extra memory and reduce the demand for labels with low-cost AL adaptations. Our results exposed quite a few reductions in terms of labels without compromising the predictive performance over a great part of the 26 datasets explored.

In Sect. 2, we introduce the concepts of Active Learning and how it is applied in streaming situations and the incremental trees. In Sect. 3, we explain how the experiments were setup and evaluated. In Sect. 4, the results of the experiments are discussed and analysed and finally, in Sect. 5, we conclude the paper and present directions for future work.

2 Active Learning and Stream Mining

Various challenges permeate data stream mining. The main ones are the volume, the velocity, the volatility of data and constraints of memory consumption. The most efficient solutions demand labels, which can pose difficulties to use the solutions within a real-life scenario. An initial work of Žliobaitė et al. [34] described some theoretical strategies to support mechanisms to control and distribute the labelling over time with balancing capabilities to induce more accurate classifiers and to detect changes. Our proposal arose from strategies and hypothesis related to Žliobaitė et al.'s contributions.

2.1 Active Learning Strategies

The core of AL strategies is composed by sampling strategies and query decision approaches, as shown in Fig. 1. The sampling strategies are different forms of directed search techniques seeking to identify samples from areas of uncertainty. On the other hand, query decision approaches regard methods to decide whether or not to query for the true label, so that the predictive model could train itself with this new instance [34].

Sampling strategies seek for areas in the input domain where the learner believes it will perform incorrect classifications, as opposed to random sampling techniques [3]. There are three main sampling strategies in the literature: membership query synthesis, stream-based and pool-based. Stream and pool-based are part of selective sampling [26].

Membership query synthesis [2] works by querying new synthesized samples based on the underlying distribution of the area of uncertainty in the input domain instead of using the already existing data. This strategy suffers from difficulty in finding methods to synthesize data instances that a human oracle is capable of interpreting. For example, when using an image dataset and interpolation for synthesizing the queried samples, the results may be a mix of different images from the input that does not mean anything to a human, hurdling the job of the oracle [23]. Selective sampling strategies were formulated to solve this problem [3] through the use of several approaches to query data from the input data to the oracle.

Pool-based selective sampling [24] usually assumes the input domain remains unchanged, contains a small number of n labelled instances and a large amount of m unlabelled instances. This method runs for various iterations until a stopping

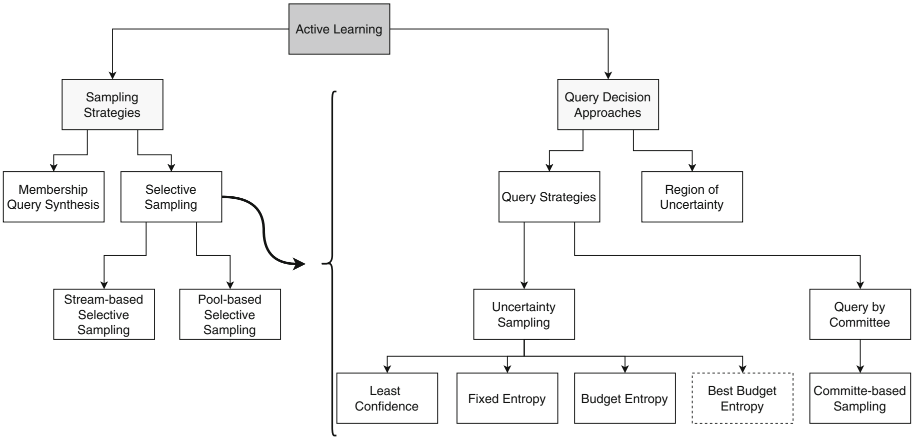

Fig. 1. Overview on the taxonomy of sampling strategies and query decision approaches. Our approach adaptation is highlighted by the dotted border.

criterion is reached, such as the oracle reaches a budget limit. Each iteration ranks the most uncertain instances, queries them to the oracle and adds them to the list of labelled instances, where the model is retrained.

Stream-based selective sampling is the most straightforward method of selective sampling. As data arrives, it will make the decision to query or not the instance to the oracle using the querying approach selected. If the learner decides to not query, that data instance is immediately discarded. This procedure can be seen in Algorithm 1.

Since we are working in a streaming scenario, the use of stream-based selective sampling is the most effective. Due to the velocity in which the data arrives, it is not feasible to use pool-based sampling due to the processes of pooling, ranking and iterations required. This same constraint limits membership query synthesis as the underlying distribution of the data needs to be analysed multiple times due to the changing nature of the data. For that reasons, in this work, we focused on stream-based selective sampling and suitable query decision approaches for a stream scenario.

Query decision approaches regards a method to decide whether or not to query for the true label so that the predictive model can train itself with this new instance [34]. The most traditional approach consists of creating an explicit region of uncertainty $R(S^m)$ where S^m is the set of m instances in the data input domain. The learner first trains on n labelled instances, where $0 < n < m$, to compute $R(S^m)$ and then simply tests each data instance for membership in $R(S^m)$, creating a collection of instances from which it will query the oracle [3]. Each new instance that falls within the region will further reduce the region when recalculated [5].

Another approach is to use query strategies to determine the most informative or uncertain data instances directly and make a decision to query them to

Algorithm 1. Stream-based selective sampling algorithm.

Input:
S: stream of unlabelled data
Q: query decision approach selected
Output:
M: A trained model
Initialize M
while *(S has next)* **do**
  $s \longleftarrow next(S)$ `// Fetch next data instance from stream`
  $o \longleftarrow M.predict(s)$ `// Get prediction outputs for` s `on` M
  if *(Q.query(o) == True)* **then**
    `// Ask query approach if the instance should be queried`
    Query s to the oracle
    Receive label for s from oracle and assign it
    Train M on s
  end
end

the oracle or not. There are many query strategies. Uncertainty Sampling [24] uses a metric to compute the uncertainty of each data instance and queries it to the oracle if it falls within a certain threshold. Committee-based sampling [9] follows the QBC [27] paradigm, with a committee of k models, where each member classifies a data instance, and the decision to query that instance is based on the classification disagreement of the members of the committee.

Uncertainty Sampling is the most used query strategy due to its simplicity. In a binary problem, for example, it would decide to query in case the probability of the predictions made by the model for either class prediction score is close to 0.5, indicating that the model is unsure as to which class the data instance belongs to. For more classes, *Least Confidence* (LC) [8] may be used. It decides to query data instances where even the class prediction with highest probability is low.

A more general and popular approach to uncertainty sampling is to use *entropy* [29] (H) (Eq. 1), where $\hat{y}_i$ is one of the labels and x^*_H is the instance to be queried to the oracle. H represents the uncertainty over the prediction output distribution with values between 0 (low uncertainty) and $log2(n\_classes)$ (high uncertainty). Technically it is an information-retrieval measure that quantifies the amount of information needed to encode the distribution [26].

$$x^*_H = argmax(H(x_i)) \text{ where } H(x) = -\sum_i P\theta(\hat{y}_i|x)log(P\theta(\hat{y}_i|x)) \tag{1}$$

A way to use entropy is by first fixing the uncertainty threshold z, and if the entropy value surpasses the specified z value, the data instance being evaluated is queried to the oracle [33]. A variant of this method is by using a budget value that limits the number of queries that can be performed. For example, a value of 0.2 means that only 20% of the instances can be labelled, in a streaming situation we can translate to something like 200 instances every 1000 [34]. We refer to this method as Budget Entropy. Budgets reflect real-life situations where the oracle has limited labeling capability and querying must be kept at a minimum.

L. Korycki et al. [19] proposes a method to decrease the number of queries made under strict budgets by using a hybrid query decision approach that uses both AL and self-labelling techniques. Self-labeling [31] is a semi-supervised learning technique that allows for the learner to label a data instance if it has a high amount of certainty on its class. This can be seen as a direct opposite of AL. This approach allows the learner to increase the number of instances used for its training with no cost. However, concept drifts are not taken in consideration and errors made by the self-labeling mechanism may propagate along the data stream.

B. Krawczyk [21] proposes a framework that is able to deal with concept drifts in limited budget situations by increasing the rate of oracle queries when drift is happening and decreasing in static situations. This framework is simple and effective, but it also has a very large amount of hyperparameters that require tuning, such as the labeling strategy and its own parameters and the adjustable rate of querying.

We proposed another method (highlighted by the dotted box in Fig. 1) grounded on entropy. Instead of a fixed uncertainty threshold, it checks the entropy of the current instance and the instance that came before, if the entropy of the current instance is higher, this instance is queried to the oracle. We call this method *Best Budget Entropy*. It has the advantages of being very simple and no hyperparameters are needed, although concept drifts are not considered directly.

In our work, we compare uncertainly sampling and stream-based selective sampling, since it is fast and effective and matches our main goal of avoiding overburdening the stream mining algorithm, particularly the incremental decision trees with an extra cost when performing AL.

2.2 Incremental Decision Trees

Hoeffding Trees [11] are incremental decision trees optimized for data stream situations. They were designed to deal with infinitely large datasets and each data instance must be read at most once in a small constant time. To achieve that, they use Hoeffding Bounds to assure that the chosen attribute for splitting with n attributes is the same as if it was chosen with infinite attributes by a margin of error ϵ. This process is done based on a function G, for example, Information Gain [25] (Eq. 2, where H is the entropy function, x the attribute and $\hat{y}$ the label), for n examples, let $G(X_1)$ be the highest value and $G(X_2)$ the second highest value among all $G(X_i)$ computed for every attribute in X and that $\Delta G = G(X_1) - G(X_2)$, for a given δ, Hoeffding Bounds guarantee that X_1 is the correct choice for the split with a probability of $1 - \delta$ if n examples were read at the node being trained and $\Delta G > \epsilon^2$.

$$IG = H(\hat{y}) - H(\hat{y}, x) \tag{2}$$

One implementation of a Hoeffding Tree is the Very Fast Decision Tree (VFDT) [11]. First, it allows choosing the G to be either Information Gain or the Gini Index. Additionally, it features a number of optimizations to further speed up the training process:

– **Tiebreak:** Tiebreak happens when two attributes have very similar values from G. Since the decision may require observation of a large number of samples to be made, this mechanism allows the learner to detect when a tie happens and simply split on the current best attribute X_i if $\Delta G < \epsilon < \tau$ for a given τ.
– **G Computation:** Since computing G can be expensive, the VFDT allows accumulation of a minimum number of samples before the G is calculated. This effectively reduces the total amount of time spent calculating G.
– **Memory Management:** In order to limit the amount of memory used, once the maximum memory available is reached, the VFDT deactivates the least promising leaves in order to free memory for new ones.
– **Disabling Poor Attributes:** Removing attributes that do not show potential, memory usage can be further minimized, this is done by dropping attributes that have a value of G with a difference of at least ϵ to the G of the best attribute.
– **Grace Period:** This allows the tree to be initialized with a small subset of data with a conventional learner, allowing the VFDT to reach better accuracies early on with a small number of samples.
– **Rescanning:** If the data arrives slowly enough or is a small finite dataset, previously observed samples can be reexamined.

SVFDT [6] is an optimization made on *VFDT*, it manages to keep a significantly lower memory footprint than the original VFDT while retaining similar predictive performance by enforcing a set of restrictions that ensure a minimum amount of uncertainty, that the leaves observe a similar number of instances and that the attributes used for the splits have relevance to the statistics.

Additionally, there is a mechanism in place that limits unnecessary growth in the tree by checking the Entropy and Information Gain values of the leaves with the other leaves and when the rules for splits in the *VFDT* were met with the Eq. 3, where X represents the observed data instances, $\overline{X}$ their mean, $\sigma(X)$ their standard deviation and x the observation of a new data instance. It is also assumed that X follows a normal distribution.

$$\varphi(x, X) = \begin{cases} True, & \text{if } x \geq \overline{X} - \sigma(X) \\ False, & \text{otherwise} \end{cases} \tag{3}$$

SVFDT is split into two versions: the *SVFDT-I* and *SVFDT-II*. Their difference consists of an additional set of constraints found in the II version that allows the node to skip all the constraints set by the growth mechanism. This set consists of two constraints that check the values of Entropy and Information

Gain for the leaves with their values for when the rules for splits in the *VFDT* were met with the Eq. 4.

$$\varpi(x, X) = \begin{cases} True, & \text{if } x \geq \overline{X} + \sigma(X) \\ False, & \text{otherwise} \end{cases} \tag{4}$$

Our approach to AL allows it to easily plug in any stream mining base learner with minimal cost as it is seamlessly integrated into the learner's input pipeline. This means that our active learning methods work as a separate module to the learner, needing only its prediction statistics to determine what instances should be queried to the oracle and feeding this data to the classifier. This can be seen in Fig. 2.

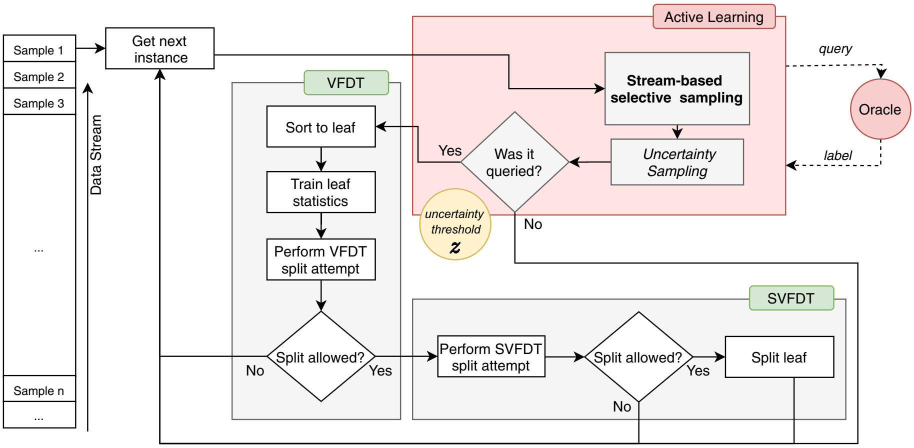

Fig. 2. Overview of stream-based selective sampling coupled to VFDT and SVFDT.

3 Experimental Setup

In this section, we present the experimental definitions to support the proposed AL method embedded into *VFDT*, *SVFDT-I* and *SVFDT-II*. To evaluate the impact of the AL methods in the trees, 26 benchmark datasets, commonly used in data stream mining experiments, were selected: – Datasets from MOA [4]: *Airlines* and *Electricity Normalized*. – Datasets from the UCI [12]: *Poker Hand* and *Covertype*. – Datasets from Weka [18]: *LED24* (with 1M instances and three files with 0%, 10% and 20% noise each) and *RandomRBF* (250k instances and 50 features, 500k instances and 10 features and 1M instances and 10 features). – Datasets from multiple sources: *CTU13* [17] (split into 13 files, one per scenario), *hyperplane* [13], *SEA* [30] and *Usenet* [16].

Prequential evaluation was employed to evaluate the algorithms [15]. Stream-based sampling was used and Uncertainty Sampling was chosen with the three entropy variants (*Entropy*, *Budget Entropy* and *Best Budget Entropy*). This was preferred over calculation of Region of Uncertainty since it is more efficient considering the streaming scenario. The *VFDT* and *SVFDTs* were compared using the parameters seen in Table 1.

Most hyperparameters were chosen with their default value (tiebreaker, split criteria, leaf prediction type and binary splits), while for the grace period we used non-default values and poor attributes are discarded to preserve memory.

Table 1. Parameter values for each incremental tree.

Parameter	*VFDT*	*SVFDT-I*	*SVFDT-II*
Split criteria	Information gain		
Grace period	100		400
Tiebreaker	0.05		
Leaf prediction type	NBAdaptive		
Only binary split	False		
Disable poor attributes	True		

We evaluated the algorithms in terms of predictive performance and queries reduction for each specific AL strategy. The predictive performance was measured using accuracy. In this work, our oracle returns the true label for the queried sample.

Two other metrics were evaluated, Relative Accuracy and Relative Query Request. Relative Accuracy is the accuracy of each AL experiment compared to the standard supervised learning accuracy while Relative Query Request is the percentage of queries made on each experiment related to the total amount of samples in each dataset.

The algorithms and AL strategies were implemented in Python 3.8. The code for this implementation can be seen in https://github.com/Vini7x/pystream-act.

4 Results and Discussions

In this section, first, we present a comparison among *VFDT*, *SVFDT-I* and *SVFDT-II* using the Relative Accuracy and Relative Query Request across several z values. Then, we perform a similar evaluation, but using each AL method across all algorithm to support generalized insights. We observed the queries rate and the impact over accuracy to discuss the trade-off between the reduction of labelling and predictive performance.

Regarding AL relative accuracy from the incremental trees, a very similar performance across four different z values (0.1, 0.2, 0.5 and 0.9) was observed, as Fig. 3 shows. Also, when compared to the usage of all samples, a slight reduction

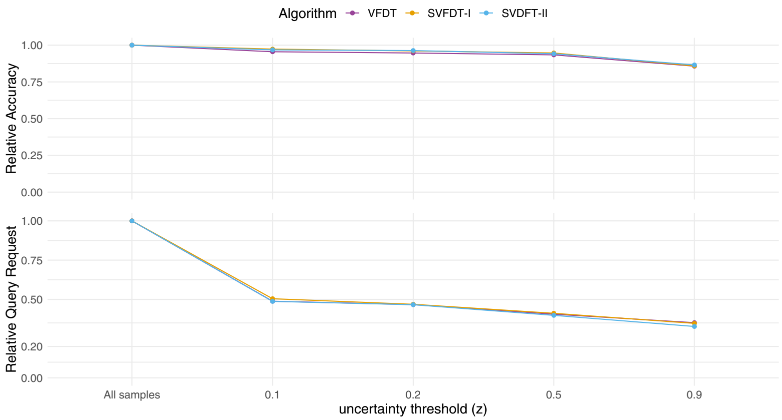

Fig. 3. Performance of different incremental trees based on relative accuracy and relative query reductions over different uncertainty threshold (z) values.

of relative accuracy was observed with the z values of 0.1, 0.2 and 0.5, respectively. A notable reduction was observed when z is equals to 0.9. On the other hand, considering the Relative Query Request, when z equals 0.1, the number of labelled samples was reduced to 49% by *VFDT* and *SVFDT-II* maintaining a low reduction in performance of about 4% and 3%, respectively. If we evaluate a trade-off using a rate of Accuracy per Query Request, *SVFDT-II* was the best combination delivering 13% of accuracy reduction using just 33% of original labelled data, as showed in Table 2.

Table 2. Table of Relative Accuracy and Relative Query request across all uncertainty threshold (z) and incremental trees.

z	Relative Accuracy			Relative Query Request		
Value	*VFDT*	*SVFDT-I*	*SVFDT-II*	*VFDT*	*SVFDT-I*	*SVFDT-II*
All samples	1.00	1.00	1.00	1.00	1.00	1.00
z = 0.1	0.96	0.97	0.97	0.49	0.50	0.49
z = 0.2	0.95	0.96	0.96	0.41	0.41	0.47
z = 0.5	0.93	0.95	0.94	0.41	0.40	0.40
z = 0.9	0.86	0.86	0.87	0.35	0.35	0.33

When evaluated from an AL perspective, we can see that the *Budget Entropy* method was the best performing of the three AL sampling strategies. Although it had the lowest accuracy of all methods, its difference was still minor while resulting in a large reduction in the number of instances queried, as can be seen in Fig. 4. Regardless of the best AL strategy, all of them were very close to the traditional supervised method in accuracy, showing that even though less data was used to train the models, the high informativeness of the queries performed by AL allowed the algorithms to reach very competitive performances.

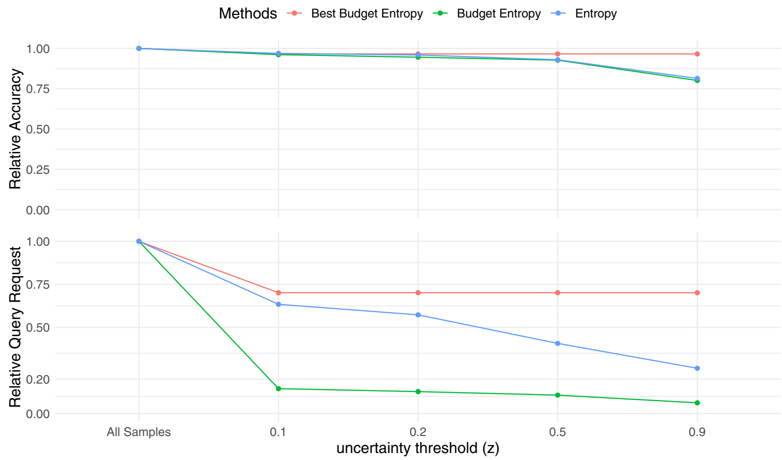

Fig. 4. Performance of AL methods based on relative accuracy and relative query reductions over different z values.

Best Budget Entropy obtained the lowest reduction in number of queries. Since it does not have the z hyperparameter, its results remain stable across all experiments. When observing Relative Query Request, *Entropy* obtained intermediate reductions as Table 3 shows. An impressive reduction was obtained by *Budget Entropy* using $z = 0.9$. It also achieved a relative accuracy of 0.80 using just 6% of available samples. This was the best trade-off between accuracy and number of query request.

Table 3. Table of Relative Accuracy and Relative Query request across all uncertainty threshold (z) and AL methods, *Best Budge Entropy* (BBH), *Budge Entropy* (BH) and *Entropy*.

z	Relative Accuracy			Relative Query Request		
Value	BBH	BH	H	BBH	BH	H
All samples	1.00	1.00	1.00	1.00	1.00	1.00
z = 0.1	0.96	0.96	0.96	0.70	0.14	0.63
z = 0.2	0.96	0.94	0.95	0.70	0.12	0.57
z = 0.5	0.96	0.92	0.92	0.70	0.10	0.40
z = 0.9	0.96	0.80	0.81	0.70	0.06	0.26

Observing the query rates across the relative accuracy intervals of 0.70, 0.75, 0.80, 0.85, 0.90, 0.95 and 1.0 in Fig. 5, it is possible to observe that *Budget*

Entropy kept the number of queries quite reduced for several accuracy intervals. The adapted approach, *Best Budget Entropy*, achieved more stability in comparison to *Entropy* in the highly accurate intervals (0.95 and 1.00).

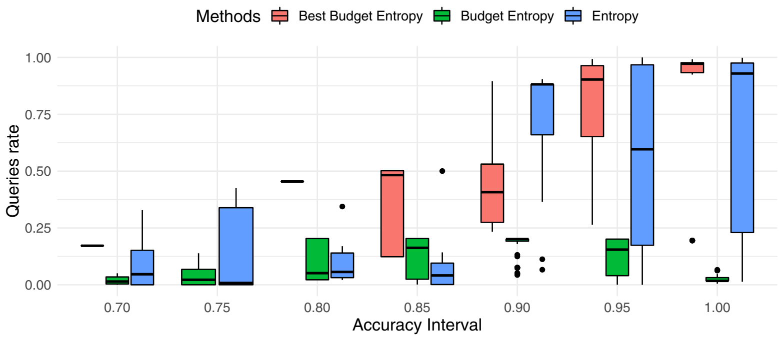

Fig. 5. Percent reduction of queries across relative accuracy interval comparing the AL methods.

Beyond the reduction of label demand, it is important to note that some accuracies obtained with AL methods surpassed the incremental decision tree results with all samples. Precisely, 81 cases distributed among algorithms, methods and some streams ($CTU13-7$, $LED24-10\%$, $Airlines$, $Usenet$, $LED24-20\%$, $CTU13-12$, $CTU13-13$, $CTU13-2$, $CTU13-9$, $CTU13-6$, $RandomRBF-1M$ and SEA). Particularly, the best improvement was about 0.8% using 99.8% of samples over $CTU13-13$ with a *SVFDT-I*. The best trade-off was achieved over $RandomRBF-1M$, in which 40.5% of samples were able to induce a model with an improvement of 0.3% over the accuracy of the model created wit all samples. The method used was Entropy. These results indicate that future work investigating alternative strategies to choose the training samples focusing on predictive performance improvements can be viable.

5 Conclusion and Future Work

We evaluated the use of three different AL methods (*Best Budget Entropy*, *Budget Entropy* and *Entropy*) in a streaming scenario for three variations of the Hoeffding Tree (*VFDT*, *SVFDT-I* and *SVFDT-II*). We observed that although regular training has higher accuracy than active learning strategies, the difference was very low in face of the amount of labelled data needed to train the model. *SVFDTs* took more advantage with the use of AL. Furthermore, in some cases *SVFDT-I* coupled with AL was able to improve the results in comparison to the use of all labelled samples. Grounded on the results, it is possible to affirm the *Budget Entropy* is the best Active Learning method to be embedded in the evaluated incremental trees. This method reached the best relation between high accuracy and reduction of query requests, since using just 14% of the total labelled samples result in only 4% of accuracy reduction.

In future work, we will explore the use of AL methods embedded in the studied incremental trees being used as base-learners into ensembles. This study will support the identification of the cost of concept drift in terms of queries required by an oracle.

References

1. Alabdulrahman, R., Viktor, H., Paquet, E.: An active learning approach for ensemble-based data stream mining. In: International Conference on Knowledge Discovery and Information Retrieval, vol. 2, pp. 275–282. SCITEPRESS (2016)
2. Angluin, D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1988)
3. Atlas, L.E., Cohn, D.A., Ladner, R.E.: Training connectionist networks with queries and selective sampling. In: Advances in Neural Information Processing Systems, pp. 566–573 (1990)
4. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. J. Mach. Learn. Res. **11**, 1601–1604 (2010)
5. Cohn, D., Atlas, L., Ladner, R.: Improving generalization with active learning. Mach. Learn. **15**(2), 201–221 (1994)
6. da Costa, V.G.T., de Leon Ferreira de Carvalho, A.C.P., Junior, S.B.: Strict very fast decision tree: a memory conservative algorithm for data stream mining. Pattern Recognit. Lett. **116**, 22–28 (2018)
7. da Costa, V.G.T., et al.: Online local boosting: improving performance in online decision trees. In: 2019 8th Brazilian Conference on Intelligent Systems (BRACIS), pp. 132–137. IEEE (2019)
8. Culotta, A., McCallum, A.: Reducing labeling effort for structured prediction tasks. In: AAAI., vol. 5, pp. 746–751 (2005)
9. Dagan, I., Engelson, S.P.: Committee-based sampling for training probabilistic classifiers. In: Machine Learning Proceedings 1995, pp. 150–157. Elsevier (1995)
10. De Rosa, R., Cesa-Bianchi, N.: Confidence decision trees via online and active learning for streaming data. J. Artif. Intell. Res. **60**, 1031–1055 (2017)
11. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Kdd, vol. 2, p. 4 (2000)
12. Dua, D., Graff, C.: UCI machine learning repository (2017). http://archive.ics.uci.edu/ml
13. Fan, W.: Systematic data selection to mine concept-drifting data streams. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 128–137 (2004)
14. Fu, Y., Zhu, X., Li, B.: A survey on instance selection for active learning. Knowl. Inf. Syst. **35**(2), 249–283 (2013)
15. Gama, J.: Knowledge Discovery from Data Streams. Chapman & Hall/CRC, 1st edn. (2010)
16. Gama, J., Kosina, P.: Recurrent concepts in data streams classification. Knowl. Inf. Syst. **40**(3), 489–507 (2013). https://doi.org/10.1007/s10115-013-0654-6
17. Garcia, S., Grill, M., Stiborek, J., Zunino, A.: An empirical comparison of botnet detection methods. Comput. Secur. **45**, 100–123 (2014)
18. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. ACM SIGKDD Explor. Newslett. **11**(1), 10–18 (2009)

19. Korycki, Ł., Krawczyk, B.: Combining active learning and self-labeling for data stream mining. In: Kurzynski, M., Wozniak, M., Burduk, R. (eds.) CORES 2017. AISC, vol. 578, pp. 481–490. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-59162-9_50
20. Krawczyk, B., Cano, A.: Adaptive ensemble active learning for drifting data stream mining. In: International Joint Conference on Artificial Intelligence (Macao), pp. 2763–2771 (2019)
21. Krawczyk, B., Pfahringer, B., Woźniak, M.: Combining active learning with concept drift detection for data stream mining. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 2239–2244. IEEE (2018)
22. Krempl, G., et al.: Open challenges for data stream mining research. SIGKDD Explor. Newsl. **16**(1), 1–10 (2014). https://doi.org/10.1145/2674026.2674028
23. Lang, K., Baum, E.: Query learning can work poorly when a human oracle is used. In: IEEE International Joint Conference on Neural Networks (1992)
24. Lewis, D.D., Gale, W.A.: A sequential algorithm for training text classifiers. In: SIGIR 1994. pp. 3–12. Springer (1994). https://doi.org/10.1007/978-1-4471-2099-5_1
25. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
26. Settles, B.: Active learning literature survey. Computer Sciences Technical Report 1648. University of Wisconsin-Madison (2009)
27. Seung, H.S., Opper, M., Sompolinsky, H.: Query by committee. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, p. 287-294. COLT 1992. Association for Computing Machinery, New York (1992). https://doi.org/10.1145/130385.130417
28. Shan, J., Zhang, H., Liu, W., Liu, Q.: Online active learning ensemble framework for drifted data streams. IEEE Trans. Neural Netw. Learn. Syst. **30**(2), 486–498 (2018)
29. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948)
30. Street, W.N., Kim, Y.: A streaming ensemble algorithm (sea) for large-scale classification. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 377-382. KDD 2001. Association for Computing Machinery, New York (2001)
31. Triguero, I., García, S., Herrera, F.: Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. Knowl. Inf. Syst. **42**(2), 245–284 (2013). https://doi.org/10.1007/s10115-013-0706-y
32. Zhu, X.J.: Semi-supervised Learning Literature Survey. University of Wisconsin-Madison Department of Computer Sciences, Technical report (2005)
33. Zliobaite, I., Bifet, A., Holmes, G., Pfahringer, B.: Moa concept drift active learning strategies for streaming data. J. Mach. Learn. Res. - Proc. Track **17**, 48–55 (2011)
34. Žliobaitė, I., Bifet, A., Pfahringer, B., Holmes, G.: Active learning with drifting streaming data. IEEE Trans. Neural Netw. Learn. Syst. **25**(1), 27–39 (2013)