# Contrary to Popular Belief Incremental Discretization can be Sound, Computationally Efficient and Extremely Useful for Streaming Data

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

***Abstract*—Discretization of streaming data has received surprisingly little attention. This might be because streaming data require incremental discretization with cutpoints that may vary over time and this is perceived as undesirable. We argue, to the contrary, that it can be desirable for a discretization to evolve in synchronization with an evolving data stream, even when the learner assumes that attribute values' meanings remain invariant over time. We examine the issues associated with discretization in the context of distribution drift and develop computationally efficient incremental discretization algorithms. We show that discretization can reduce the error of a classical incremental learner and that allowing a discretization to drift in synchronization with distribution drift can further reduce error.**

## I. Introduction

It is surprising that discretization of numeric data streams has received little attention. One reason may be that the cut points are likely to have to change as the stream progresses, because the distribution of values may vary. This may bias potential users against using discretization because it may seem unintuitive to use discretized values whose meaning changes over time. We argue, to the contrary, that changing over time the cut points associated with each discretized value might sometimes be necessary if the interval is to retain the relevant meaning for a given task.

This paper investigates discretization of numeric stream data. We present two efficient and effective incremental discretization algorithms. The first approximates equal frequency discretization over the entire stream to the time step. The second uses a window of recent values and performs equal frequency discretization on these, allowing the cut points to exactly track a non-stationary distribution. Our experiments demonstrate that discretization can reduce error for the state-of-the-art streaming learner Logistic Regression (LR) with Stochastic Gradient Descent. We further demonstrate that for some streaming data it is indeed useful to have discretizations whose cut points change over time, tracking the evolution of the underlying concepts.

## II. Discretization for streaming data

We wish to incrementally update a model $\Theta$ to predict the posterior probability distribution $P(y \mid \mathbf{x}_i)$ of the classes $y_i \in \{c_1, \dots c_k\}$ for objects $\mathbf{x}_i = \langle x_i^1, \dots, x_i^a \rangle$ while viewing a large or infinite stream $\mathcal{S} = \{\mathbf{x}_1, \dots \mathbf{x}_n\}$ of objects. We use $\Theta_i$ to denote the model at time step $i$ and $P_{\Theta_i}(y \mid \mathbf{x}_i)$ to denote the class distribution predicted by model $\Theta_i$ for object $\mathbf{x}_i$. We assume that the true class $y_i$ for each $\mathbf{x}_i$ becomes available after $\mathbf{x}_i$ is classified and can be used for subsequent training of the classifier. The attribute values $x_i^j$ of the objects may be either categorical or numeric.

A discretization $\delta$ of a numeric attribute $X_i$ is a set of $m$ intervals called *bins*. These bins can be defined by cut points $\{\kappa_1, \dots, \kappa_{m-1}\}$. These cut points divide the domain of $X_i$ into bins $b_1 \dots b_m$ using a scheme such as $b_1 = [-\infty, \kappa_1], b_m = (\kappa_{m-1}, \infty]$ and for $1 < i < m, b_i = (\kappa_{i-1}, \kappa_i]$. A discretization of attribute $X_i$ defines a mapping between values $v$ of $X_i$ and bin indexes, $\delta v = z$ such that $v \in b_z$.

Discretization is closely related to both *histograms* and *quantiles*. A histogram of a numeric attribute $X_i$ with respect to a dataset $\mathcal{S}$ can be viewed as a discretization of $X_i$ augmented with a vector of counts $\eta_1, \dots \eta_m$ such that $\eta_k$ represents $|\{j : x_j^i \in b_k\}|$, the number of records whose value for the attribute falls within the bin.

A $p^{\text{th}}$ quantile $Q_i^p$ of an attribute $X_i$ with respect to $\mathcal{S}$ is a value such that $|\{j : x_j^i < Q_i^p\}|/n < p \wedge |\{j : x_j^i > Q_i^p\}|/n < 1 - p$. That is, it is the value of $x_{pn}^i$ if the data were sorted on the attribute. If $pn$ is not an integer then the $p^{\text{th}}$ quantile may be any value in $[x_{\lfloor pn \rfloor}^i, x_{\lceil pn \rceil}^i]$ and is often set to $x_{\lfloor pn \rfloor}^i + (x_{\lceil pn \rceil}^i - x_{\lfloor pn \rfloor}^i)/2$.

## III. Issues in discretization for streaming data

The cut points for discretization of streaming data may need to change over time. This is because the process that generates the stream $\mathcal{S}$ may be non-stationary, in which case it is not going to be possible to anticipate what the future distribution of values for an attribute will be and hence impossible to predetermine what intervals will be relevant in the future. If the intervals are predetermined and remain static then they are likely to eventually lose relevance. However, such changes to the intervals over time may appear undesirable, as they seem to imply that the meanings of the bins must change. We suspect that this has been a key reason why there has been little previous research into discretization for streaming data.

However, this concern may be misguided. If a distribution is non-stationary then it actually may be desirable for the discretization to drift in synchronization with the changes in the distribution. For example, consider a stream of data that includes an *income* attribute. The values of this attribute can be

Proceedings 14th IEEE International Conference on Data Mining, pp. 1031-1036, 2014. Updated 24 Nov 2016 to fix typo in Table II and the description of the InsertValue algorithm.

expected to grow over time. For at least some applications it seems credible that we should want the discretization to reflect this evolution. For example, it may be necessary for the cut point on a bin representing *high income* to increase over time if that interval is to retain its relevant meaning.

A further issue is that some algorithms do not require continuity over time in the bins that are used. For example, Naive Bayes [1] requires at classification time estimates of the prior probability of each class, $P(y_i)$ and of the likelihood of each attribute value given the class $P(x_i^j \mid y_i)$. These can be derived from counts of the relative frequency of each class and of each pair of class and attribute value. It is not relevant what the intervals were for previous classifications, only that these necessary statistics be available for the current discretization. Hence, Naive Bayes can be well served by a technique that maintains a suitable augmented histogram over time and it is irrelevant whether the number of bins or their cut points change. Rather, the key issue is whether the counts are sufficiently accurate for effective classification [2].

On the other hand, most discriminative learning algorithms do not operate in this manner and do require that the number of bins and their meaning be constant over time. For such algorithms, quantile-based discretization, such as equal frequency discretization, may be effective. This unsupervised discretization strategy requires that the number of bins, $m$, be pre-specified, together with a set of quantiles that specify the cutpoints. For equal frequency discretization the range of attribute $X_i$ is divided into $m$ bins, each containing the same number of training examples, that is, into bins $b_1, \dots b_m$ such that $\forall k, l \in (1, m)\, |\{j : x_j^i \in b_k\}| = |\{j : x_j^i \in b_l\}|$. This is directly related to the problem of finding quantiles, as the $k^{\text{th}}$ bin has an interval $(Q_i^{\frac{k-1}{m}}, Q_i^{\frac{k}{m}}]$.

Quantile-based discretization allows at least one type of meaning of an interval to remain invariant even while the cut points change. Consider again the case of an attribute for income. Suppose it is discretized into three bins, the lower, middle and upper thirds of income. If a streaming discretization algorithm is able to maintain such a discretization over time, varying the cut points as needed, at least one potentially important meaning of the intervals will remain constant.

Supervised discretization often results in more useful bins than unsupervised approaches [3]. However, supervised discretization does not appear feasible for discriminative learners in a streaming context, as the cuts selected by a supervised approach may vary dramatically over time and classical discriminative learners cannot track and adjust for this. In contrast quantile-based discretization can maintain a constant set of bins, each with a meaning that remains invariant even while the cut values that define the bins drift. If meaningful quantiles can be identified for a learning problem then these should be used. However, we show that even when such information is not known, simple equal frequency discretization can be effective.

It may appear counter-intuitive that discretization should improve the performance of a learning algorithm that can handle numeric values directly, because it is clear that discretization loses information. However, even though a discretized variable contains less information than the undiscretized original, the models that a learner forms may be able to employ that information more effectively.

TABLE I. UPDATE SAMPLES

```
globals
$s$: the sample size
$n$: the number of instances seen in the stream to date
$V$: a vector of set of samples, indexed by attribute
 1: procedure UPDATESAMPLES($\mathbf{x} = \langle x_1, \dots, x_a \rangle$)
 2:     if rand() $\le s/n$ then
 3:         for $i$ = 1 to $a$ do
 4:             if $x_i$ is not missing then
 5:                 if $|V_i| = s$ then
 6:                     remove a random element from $V_i$
 7:                 end if
 8:                 add $x_i$ to $V_i$
 9:             end if
10:         end for
11:     else
12:         for $i$ = 1 to $a$ do
13:             if $|V_i| < s$ and $x_i$ is not missing then
14:                 add $x_i$ to $V_i$
15:             end if
16:         end for
17:     end if
18: end procedure
```

Consider for example a simple linear model such as created by Logistic Regression. Such a model requires that the predictiveness of a numeric value be proportional to its value. It cannot directly model the case where only unusually high values are indicative of one class, and average or low values are all equally indicative of the other, or where average values are indicative of one class and either high or low values indicative of the other. By discretizing the attribute and then treating each discrete value as a binary variable a linear classifier can model the predictiveness of individual segments of the number line irrespective of their relative absolute values.

## IV. INCREMENTAL DISCRETIZATION ALGORITHM IDA

The Incremental Discretization Algorithm (IDA) approximates quantile-based discretization on the entire data stream encountered to date by maintaining a random sample of the data which is used to calculate the cut points.

A random sample is used because: 1) it is not feasible for high-throughput streams to maintain a complete record of all values observed to date; 2) it is computationally efficient; and 3) it is possible to place tight bounds on the expected variance of the cut points [4].

We use the reservoir sampling algorithm [5] to maintain the random sample of $s$ values $V_i$ for each attribute. The first $s$ values of each $X_i$ are added to the corresponding $V_i$. Thereafter, when the $n^{\text{th}}$ object $\langle \mathbf{x}_n, y_n \rangle$ is encountered, with probability $s/n$, each of its values $\mathbf{x}_n^i$ replaces a randomly selected value of the corresponding $V_i$. See Table I.

We store the values of each attribute in a vector of interval heaps [6], where $V_i^j$ stores the values for the $j^{\text{th}}$ bin of $X_i$. This provides efficient access to the minimum and maximum values in a bin, and direct access to a random value within a bin when replacing a value selected at random. This data structure ensures that insertion and deletion are of order $O(\log s)$ and retrieving a cut point is constant time. The algorithm for inserting a value $v$ into $V_i$ is presented in Table II. Recall that

TABLE II. INSERT VALUE

```
globals
$m$: the number of bins
 1: procedure InsertValue($v$, $V_i$)
 2:     $t = |V_i|$ mod $m$
 3:     $j = \mathbf{argmin}_j \uparrow V_i^j \geq v$
 4:     insert $v$ into $V_i^j$
 5:     if $j < t$ then
 6:         for $k = j$ to $t - 1$ do
 7:             add $\uparrow V_i^k$ to $V_i^{k+1}$
 8:             remove $\uparrow V_i^k$ from $V_i^k$
 9:         end for
10:     else
11:         for $k = t$ to $j - 1$ do
12:             add $\downarrow V_i^{k+1}$ to $V_i^k$
13:             remove $\downarrow V_i^{k+1}$ from $V_i^{k+1}$
14:         end for
15:     end if
16: end procedure
```

$m$ is the number of bins. $\uparrow V_i^j$ and $\downarrow V_i^j$ denote, respectively the maximum and minimum value in $V_i^j$. Line 2 finds the target bin — the next bin that should increase in size. Line 3 uses binary search to find the bin in which the value belongs. The value is inserted into the appropriate bin. If it is not the target bin, the excess value is shuffled up or down to the target.

Deletion is a minor variation on insertion. The cut points are accessed in constant time by returning the maximum value of the appropriate bin.

IDA maintains a random sample of the stream from its beginning to the current point of time. As suggested in the introduction, in some contexts it might be valuable to have the intervals drift, so that the actual cut point associated with the lowest range of income, for example, drifts upwards as inflation increases incomes. IDA's intervals will drift over time to reflect overall changes in the total distribution to date. However, it does not directly track the current distribution. A variant that more precisely tracks the evolution of a data stream is to maintain $S$ as a window of the $s$ most recent objects. In this case the discretization will change as the distribution changes, but will be more subject to frequent random minor fluctuations than a more gradual update approach. We call the latter approach the Incremental Discretization Algorithm with a Window (IDAW). This requires the additional overhead of maintaining for each value the window of values in time order so that the oldest value can at each step be identified and replaced by the newest value.

## A. Computational Complexity

The computational complexity of IDA is dominated by the costs of maintaining the samples and determining the quantiles from those samples. The required operations are to insert a new value (only required while the sample is not yet at full size), to replace a random value with a new value, and to return the required quantiles.

As each bin is maintained as an interval heap [6], finding the quantiles takes constant time and inserting or removing a value from a bin $V_i^j$ takes $O(\log |V_i^j|) = O(\log(s/m))$ time. As replacement requires up to $m$ insertions and deletions, replacement requires order $O(m \log(s/m))$ time.

However, these relatively expensive updates are only required on average once every $s/t$ updates, where $t$ is the current time step or size of the stream to date. Thus the amortized cost is $O([\sum_{i=1}^{s} m \log i/m + \sum_{i=s+1}^{t} \frac{s}{i} m \log s/m]/t)$, where the first term represents the initial $s$ time steps during which the sample is built up to its operating size and the second term represents updates to the sample once it reaches operating size. It is readily apparent that these updates rapidly become very rare and that as the size of the stream becomes very large the amortized cost becomes negligibly small.

The situation is more complex for IDAW, which maintains a window of the $s$ most recent values for each attribute. This requires that the values be maintained in both time and value order. Maintaining an order by time can be achieved very efficiently with a circular buffer, which supports all updates and accesses in constant time. As the elements to be replaced in a replacement operation are no longer selected at random, it is not efficient to maintain the bins as interval heaps, as above. Rather we need to use slightly more expensive balanced binary trees for which the time to identify the location of the value to be removed is $O(\log(s/m))$, which this does not increase the overall complexity of the update operation relative to that for IDA. The major computational penalty, however, is that these updates must be performed for every object encountered in the queue, which makes the maintenance of the discretization a non-trivial ongoing overhead.

## V. Related Research

As we have noted above, maintaining the $i/m$-quantiles for each $1 \leq i \leq m$ is the key requirement in order to discretize a data stream into $m$ equal frequency bins. These quantiles provide the required cut points. Algorithms exist for finding approximate quantiles in data streams with strict bounds on the error [7] and [8]. However, they rely on the records in the stream appearing in random order, a requirement that is likely to be strongly violated in many learning applications. This renders these algorithms inappropriate for our purposes.

A discretization technique should be matched to the properties of the learning algorithm. A number of papers have investigated discretization of streaming data in the context of naive Bayes (NB) [9]–[11]. NB is an unusual algorithm in that the model it learns for categorical data can be represented in the form of an augmented histogram, requiring counts of both the frequency of each attribute value and the joint frequency of each combination of an attribute and a class value. As a consequence it does not matter if there is a change in either the number of values of an attribute or the meaning of an attribute value, so long as the appropriate counts are maintained. In contrast, many other incremental learning algorithms, such as linear classifiers with weights learned by stochastic gradient descent, require that the number of attribute values remains constant and that their meanings do not change. In the current work we target algorithms that require the number of bins and their meanings to be invariant.

Partition Incremental Discretization (PID) [10] allows the number of intervals to remain constant. It operates by forming two layers of discretization. The top layer is the discretization used by the learning algorithm. The bottom layer contains many more bins than the top layer. In their example case for equal frequency discretization the bottom layer aims to maintain bins that contain $1/20$ the number of instances required by each bin at the top level. Top level bins are formed by aggregation of consecutive lower-level bins until approximately the correct size bin is obtained. The lower-level bins are initially formed by setting cut points at equal distances along the number line between an indicative lower and upper value on the attribute. Then as the stream is consumed, the counts for the lower-level bins are incremented as appropriate. When a lower-level bin exceeds a threshold size it is split on a value mid-way between its minimum and maximum values, and each count is set to one half the count for the original bin. This may result in some inaccuracy in the counts, but such inaccuracy only matters when the two parts of a split lower-level interval end up in different top-level bins, as otherwise both of the bins that have been formed will fall within the one top-level bin and the top-level bin's total count will remain accurate. The paper does not specify the threshold for splitting. In our study we use twice the target size for a lower-level bin. In other words, a lower-level bin is split in two when it exceeds 1/10$^{\text{th}}$ the target size for an upper-level bin.

PID has three potential limitations. First, as lower-level bins move from one higher-level bin to another, there might be abrupt changes in the cut points from one update to the next. Second, if the spread of values on the number line is not uniform, the number of bins created may become very large. This is because a small number of initial bins may need to be repeatedly split to accommodate the majority of the data. Third, the splitting process might result in major inaccuracies in the estimated counts when there are very large numbers of repetitions of a single value $v$. In this case the lower-level bin into which $v$ falls will rapidly grow to exceed the size threshold and be split. However, the division of the counts across the two resulting bins will be inaccurate, as all the repetitions of $v$ belong in the same bin but will be attributed equally to each of the new bins. This may occur repeatedly, causing a diminishingly small proportion of the true count for $v$ to be allocated to the correct bin.

## VI. Evaluation

We seek to evaluate three primary contributions — 1) *IDA*, a new algorithm for efficient and effective discretization of streaming data that approximates the maintenance of equal frequency discretization over all of the data observed up to the current time; 2) *IDAW*, a variant of IDA that seeks to maintain an equal frequency discretization over the data distribution at the current time; and 3) the hypothesis that discretization based on quantiles can allow the cut points to drift over time without changing the relevant meaning of the intervals. It is also important to understand exactly how much power is lost by performing incremental rather than batch discretization. To assess these contributions and issues we evaluate each component of our new algorithms in turn.

We first compare discretization using the full data (Pre-Disc) against discretization using all the data encountered up to the time of classification (All-So-Far). Note that both Pre-Disc and All-So-Far set hypothetical benchmarks. Neither is feasible in a real-world streaming data context because the first requires seeing all data that will ever come through the stream in advance and the second requires retaining and analyzing all data in the stream.

The next relevant test is to assess the loss in accuracy that results from using a random sample rather than discretizing on all the data encountered to date. To this end we compare IDA against All-So-Far.

It is also important to compare against the current state-of-the-art in incremental discretization, PID. This is the only prior incremental discretization technique capable of supporting equal frequency discretization. PID requires that the user provides an initial estimate of the likely minimum and maximum value for each attribute. To ensure that the evaluation is as favorable to PID as possible we use the true minimum and maximum in place of these estimates, values that are often not known in practice for real streaming data.

In order to understand what advantage, if any, discretization can confer, we compare LR with IDA to LR performed on normalized numeric data (No-Disc). To ensure that this comparison is as favorable as possible to the non-discretization option we normalize using the minimum and maximum values of each attribute in the data, replacing each value $x_i$ with

$$2(x_i - {\downarrow}X_i)/({\uparrow}X_i - {\downarrow}X_i) - 1.0, \tag{1}$$

where ${\downarrow}X_i$ and ${\uparrow}X_i$ denote respectively the minimum and maximum values for the attribute $X_i$. This normalizes values to the interval $[-1.0, 1.0]$. Such normalization would clearly often not be possible in practice with streaming data because it is often not possible to know in advance the minimum and maximum values for an attribute.

We also wish to investigate the idea of allowing the discretization to drift over time, closely tracking the current distribution of values. To this end we compare IDAW to IDA using sample sizes of 1000.

We perform all experiments using LR with single-pass Stochastic Gradient Descent (LRSGD) using regularization rate $\mu = 0.001$ and learning rate or step size $\lambda = 0.001$. These are rates that we have found to be effective in previous experimental work on the current datasets when not using discretization. The regularization rate is not reduced over time as we are seeking to learn in the presence of distribution and concept drift and hence the target is non-stationary and so we cannot assume that we are ever approaching a fixed optimum. LRSGD has been selected as an exemplar of the type of incremental learning algorithm normally associated with numeric data that we believe may benefit from discretization.

All experiments use the procedure outlined in Table III. We use 5 bin discretization because 10 bins obtained the same overall results and 5 bins provided the best results for Pre-Disc.

### A. *Comparisons without distribution or concept drift*

We are presenting a new approach to discretization. While it is designed for use with streaming data it is important to establish how much accuracy is lost relative to the non-streaming baseline in a situation where there is clearly no distribution or concept drift. To this end we perform experiments where the data are shuffled to ensure there is no systematic drift

TABLE III. STREAM LEARNING PROCEDURE

```
1: procedure StreamTest(data stream: $\mathcal{S}$,
                         discretizer: $\Delta$,  learner: $\lambda$)
2:     initialize the discretization $\delta$ as required by $\Delta$
3:     for i = 1 to —$\mathcal{S}$— do
4:         update $\delta$ by applying $\Delta(\delta, \mathbf{x}_i)$
5:         apply the learner, $\hat{y} = \lambda(\delta(\mathbf{x}_i))$
6:         record the error $\mathrm{I}(\hat{y} \neq y_i)$
7:     end for
8: end procedure
```

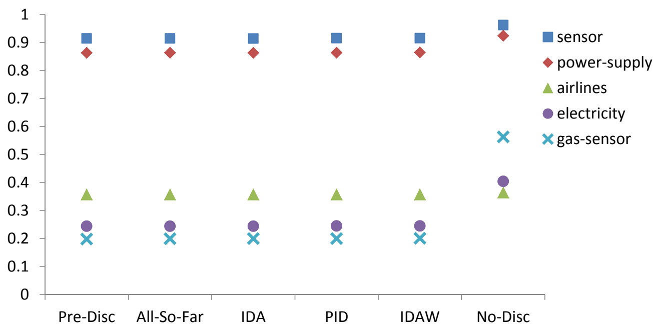

Fig. 1. 0-1 loss on data without distribution drift

over time and compare our streaming algorithms against pre-discretization using all the data and a streaming discretization that uses all the data up to the current point of time. 20 experiments were performed for each data stream, each time shuffling the data in advance.

We use the only public real-world stream classification datasets of which we are aware, airlines and electricity, obtained from the MOA website [12]; gas-sensor, obtained from the UCI Repository [13]; and power-supply and sensor, obtained from the Stream Data Mining Repository [14]. We present the resulting mean 0-1 loss for each algorithm on each dataset in Fig. 1, with error bars representing 1 standard deviation marked, but too close to be readily discerned.

We use two-tailed match-pair t-tests for significance, employing an adjusted critical value of $0.05/75 = 0.000\dot{6}$ after a Bonferonni correction for the 75 comparisons performed (15 pairs of algorithms times 5 datasets). In no case is there a significant difference between the error of the discretization techniques on airlines, electricity, gas-sensor or power-supply ($p = 0.0012$ to $0.7931$). On sensor, IDAW has significantly higher error than the other discretization techniques ($p = 1.39 \times 10^{-13}$ to $9.33 \times 10^{-06}$ ). This may be due to the instability of the quantiles as they are continually updated. On all streams the use of LR without discretization results in higher error than its use with any discretization technique ($p = 1.92 \times 10^{-39}$ to $4.96 \times 10^{-28}$).

These results demonstrate that our computationally efficient use of small samples provides performance that is close to optimal in the absence of concept drift.

## B. Comparisons on real world data

To establish the value of our algorithms in the context of distribution drift, it is useful to assess performance on real-world stream data. Our final study compares the algorithms on the real-world data used in the previous experiment.

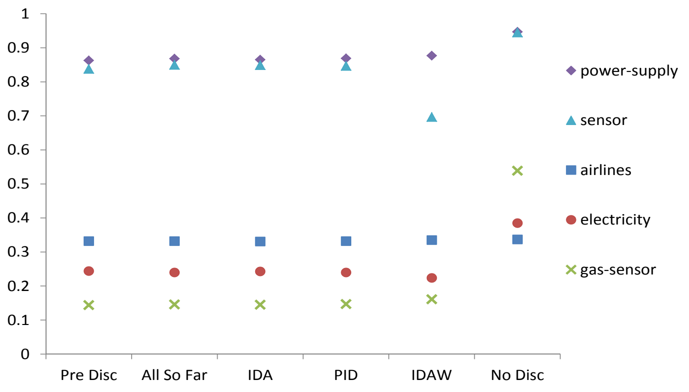

Fig. 2. Errors for each approach on each data stream

We process our real-world datasets in their original order. Because they are in a fixed order it is not possible to have repeated trials and hence not possible to perform statistical tests. In consequence, one should be cautious in interpreting the apparent differences as meaningful unless they are quite substantial. The 0-1 loss is presented in Figure 2.

All the discretization techniques appear to enjoy a substantial advantage relative to no discretization on all data streams other than airlines for which the advantage is small.

Maintaining an exact discretization over all the data to the current point offers similar accuracy to pre-discretization on all datasets except sensor for which it appears to substantially increase error. It is not apparent why All-So-Far should be penalized on this particular data stream.

The two approaches that seek to approximate All-So-Far, IDA and PID, both achieve error very close to its error.

The IDAW approach of tracking the current distribution delivers very substantial reductions in error for the electricity and sensor data streams, but results in substantial increases in error for gas-sensor and power-supply. The benefit of this approach on the electricity and sensor data streams supports our hypothesis that maintaining discretizations based on quantiles as they vary over time can maintain meaning while the cut-points vary. However, the results for the other data streams show that some types of distribution drift do not take this form.

## C. Running times

Due to the large number of repetitions of processing large datasets we conducted all experiments on a heterogeneous grid system. As a result, compute times are only indicative at best. Nonetheless we present in Figure 3 the compute times for the experiments on real-world data in order to give a feel for the computational profiles of the techniques that we have developed. The software is implemented in C++ but little attempt has been made to optimize the discretization process.

The key observation is that IDA and its variants in most cases incur only modest computational overheads relative to no discretization. The relatively poor performance of PID should be treated with caution as we have made no attempt to optimize our reimplementation of the technique.

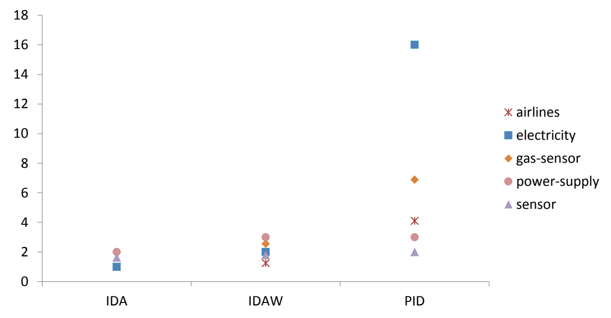

Fig. 3. Running times for each incremental discretization technique on each data stream, presented in multiples of time taken without discretization

## VII. Conclusions

We have explored the key issues that surround discretization of streaming data and presented two new techniques based on sampling. Most discriminative algorithms require that the number and meaning of the bins remain invariant. We argue that binning on fixed quantiles of the distribution, rather than fixed absolute values, can maintain an appropriate meaning over streaming data with distribution drift. Hence one bin can represent the top $p$ values for the data and so on, even as the absolute values in that range vary.

Our new stream discretization techniques use a sample of values for an attribute to maintain an equal frequency discretization. They differ only in the composition of the sample. IDA uses the reservoir sampling algorithm to maintain a sample drawn uniformly at random from the entire stream up until the current time. This approximates the maintenance of an equal frequency discretization over the entire stream up to the current point. Its desirable features include involving negligible computation once the stream becomes large, as updates to the sample become very rare. Our results show that it is very effective in the absence of concept drift and can substantially reduce the error of LRSGD. Even with a very small sample it only increases the error very modestly compared with equal frequency discretization over all data in the stream to date.

IDAW is a variant of IDA that is useful when it is desirable to more closely track the current distribution of the data. IDAW maintains a window of the most recent values for an attribute and discretizes these. This approach incurs greater computation than IDA, as the sample must be updated at every time step. Further, the values must be maintained in two orders, value order to support discretization and time order to allow maintenance of the window. Nonetheless we show that this additional computational burden can deliver substantial benefit in the context of incremental concept drift. It remains an open topic for future research whether it is possible to identify when drifting discretization such as that provided by IDAW is appropriate and when non-drifting discretization such as provided by IDA will be more effective.

The computational burden of IDAW could be greatly reduced in contexts where the rate of expected drift relative to the rate at which objects arrive is low, by only updating with occasional randomly selected objects.

We conducted our experiments using LRSGD. We have shown that with this learner discretization can deliver substantial reductions in error relative to learning from undiscretized data. This is not to claim that more sophisticated treatment of undiscretized data could not achieve even better results. Our objective is to show that discretization is a practical addition to the streaming data toolbox which is worthy of consideration, rather than to argue that it provides universal benefit.

While our research has only considered classifciation learning from stream data, discretization is likely to also prove valuable for other data mining activities on data streams including itemset mining [15] and clustering [16]. We leave it to future research to explore the potential benefits of discretization in these contexts and the relative merits of alternative stream discretization strategies.

It is a surprising gap in the data mining literature that relatively little has been done on discretization for streaming data. Perhaps the greatest contribution of this paper is to have shown that it can be done in a computationally efficient manner and that it can deliver substantial value.

The executable binaries, scripts, datasets and instructions required to replicate the experiments can be downloaded from http://www.csse.monash.edu.au/~webb/Software/incremental-discretization.tgz.

## References

[1] G. I. Webb, "Naive Bayes," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Springer, 2011, pp. 713–714.

[2] Y. Yang and G. I. Webb, "Discretization for naive-Bayes learning: Managing discretization bias and variance," *Machine Learning*, vol. 74, no. 1, pp. 39–74, 2009.

[3] S. Garcia, J. Luengo, J. Saez, V. Lopez, and F. Herrera, "A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 4, pp. 734–750, April 2013.

[4] A. Stuart and J. K. Ord, *Kendall's Advanced Theory of Statistics*, 6th ed. Edward Arnold, 1994.

[5] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Mathematical Software*, vol. 11, no. 1, pp. 37–57, 1985.

[6] J. van Leeuwen and D. Wood, "Interval heaps," *The Computer Journal*, vol. 36, no. 3, pp. 209–216, 1993.

[7] S. Guha and A. McGregor, "Stream order and order statistics: Quantile estimation in random-order streams," *SIAM Journal on Computing*, vol. 38, no. 5, pp. 2044–2059, 2009.

[8] A. Gupta and F. X. Zane, "Counting inversions in lists," in *Proc. Fourteenth Annual ACM-SIAM Symp. Discrete Algorithms*, ser. SODA '03, 2003, pp. 253–254.

[9] T. Elomaa and P. Lehtinen, "Maintaining optimal multi-way splits for numerical attributes in data streams," in *PAKDD08*, 2008, pp. 544–553.

[10] J. Gama and C. Pinto, "Discretization from data streams: applications to histograms and data mining," in *Proc. 2006 ACM Symp. Applied Computing*. ACM, 2006, pp. 662–667.

[11] J. Lu, Y. Yang, and G. I. Webb, "Incremental discretization for naive-Bayes classifier," in *Proc. 2nd Int. Conf. Advanced Data Mining and Applications (ADMA 2006)*. Springer, 2006, pp. 223–238.

[12] "MOA," 2014. [Online]. Available: http://moa.cms.waikato.ac.nz/

[13] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[14] X. Xu, "Stream data mining repository," 2010. [Online]. Available: http://www.cse.fau.edu/~xqzhu/stream.html

[15] N. Jiang and L. Gruenwald, "Cfi-stream: mining closed frequent itemsets in data streams," in *ACM SIGKDD-06*. ACM, 2006, pp. 592–597.

[16] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases-Volume 29*, 2003, pp. 81–92.