Discretization from Data Streams: Applications to Histograms and Data Mining

João Gama[1] and Carlos Pinto[2]

[1] LIACC, FEP University of Porto
R. de Ceuta, 118, 6,Porto - Portugal,
jgama@fep.up.pt
[2] University of Algarve- Portugal
cpinto@ualg.pt

Abstract. In this paper we propose a new method to perform incremental discretization. The basic idea is to perform the task in two layers. The first layer receives the sequence of input data and keeps some statistics on the data using much more intervals than required. Based on the statistics stored by the first layer, a second layer creates the final discretization. The proposed architecture process streaming examples in a single scan, in constant time and space even for infinite sequences of examples. We experimentally demonstrate that incremental discretization is able to maintain the performance of learning algorithms in comparison to a batch discretization. The proposed method is much more appropriate in incremental learning, and in problems where data flows continuously as in most of recent data mining applications.

1 Introduction

Discretization of continuous attributes is an important task for certain types of machine learning algorithms. In Bayesian learning discretization is the most common approach when data is described by continuous features [6]. Although discretization is a well-known topic in data analysis and machine learning, most works refer to a batch discretization where all the examples are available for discretization. Few works refer to incremental discretization [5, 11, 12]. This is the main motivation for this work.

The paper is organized as follows. The next section presents a new algorithm to continuously maintain histograms over a data stream. In Section 3 we extend the algorithm for predictive data mining. Section 4 presents preliminar evaluation of the algorithm in benchmark datasets and one real-world problem. Last section concludes the paper and presents some future research lines.

2 Histograms

Histograms are one of the most used tools in exploratory data analysis. They present a graphical representation of data, providing useful information about

the distribution of a random variable. A histogram is visualized as a bar graph that shows frequency data. The basic algorithm to construct an histogram consists of sorting the values of the random variable and place them into *bins*. Then we count the number of data points in each bin. The height of the bar drawn on the top of each bin is proportional to the number of observed values in that bin.

A histogram is defined by a set of non-overlapping intervals. Each interval is defined by the boundaries and a frequency count. In the context of open-ended data streams, we never observe all values of the random variable. For that reason, and allowing consider extreme values and outliers, we define an histogram as a set of break points $b_1, ..., b_{k-1}$ and a set of frequency counts $f_1, ..., f_{k-1}, f_k$ that define k intervals in the range of the random variable:
$]-\infty, b_1],]b_1, b_2], ...,]b_{k-2}, b_{k-1}],]b_{k-1}, \infty[$.

The most used histograms are either *equal width*, where the range of observed values is divided into k intervals of equal length $(\forall i, j : (b_i - b_{i-1}) = (b_j - b_{j-1}))$, or *equal frequency*, where the range of observed values is divided into k bins such that the counts in all bins are equal $(\forall i, j : (f_i = f_j))$.

When all the data is available, there are exact algorithms to construct histograms. All these algorithms require a user defined parameter k, the number of bins. Suppose we know the range of the random variable (domain information), and the desired number of intervals k. The algorithm to construct *equal width* histograms traverse the data once; whereas in the case of *equal frequency* histograms a sort operation is required. One of the main problems of using histograms is the definition of the number of intervals. A rule that has been used is the Sturges' rule: $k = 1 + log_2 n$, where k is the number of intervals and n is the number of observed data points. This rule has been criticized because it is implicitly using a binomial distribution to approximate an underlying normal distribution[1]. Sturges rule has probably survived as long as it has because, for moderate values of n (less than 200) produces reasonable histograms. However, it does not work for large n. In exploratory data analysis histograms are used iteratively. The user tries several histograms using different values of the number of intervals, and choose the one that better fits his purposes.

2.1 The Partition Incremental Discretization

The *Partition Incremental Discretization* algorithm (*PiD* for short) is composed by two layers. The first layer simplifies and summarizes the data; the second layer construct the final histogram.

The first layer is initialized without seeing any data. The input for the initialization phase is the number of intervals (that should be much greater than the desired final number of intervals) and the range of the variable. The range of the variable is only indicative. It is used to initialize the set of breaks using a

[1] Alternative rules for constructing histograms include Scott's (1979) rule for the class width: $h = 3.5sn^{-1/3}$ and Freedman and Diaconis's (1981) rule for the class width: $h = 2(IQ)n^{-1/3}$ where s is the sample standard deviation and IQ is the sample interquartile range.

```
Update-Layer1(x, breaks, counts, NrB, alfa, Nr)
x - observed value of the random variable
breaks - vector of actual set of break points
counts - vector of actual set of frequency counts
NrB    - Actual number of breaks
alfa   - threshold for Split an interval
Nr     - Number of observed values

 If (x < breaks[1]) k = 1; Min.x = x
 Else If (x > breaks[NrB] k = NrB; Max.x = x
 Else k = 2 + integer((x - breaks[1]) / step)

 while(x < breaks[k-1]) k <- k - 1
 while(x > breaks[k]) k <- k + 1

 counts[k] = 1 + counts[k]
 Nr = 1 + Nr
 If ((1+counts[k])/(Nr+2) > alfa)  {
      val = counts[k] / 2
      counts[k] = val
      if (k == 1) {
        breaks = append(breaks[1]-step, breaks)
        counts <- append(val,counts)
      }
      else {
        if(k == NrB) {
          breaks <- append(breaks, breaks[NrB]+step)
          counts <- append(counts,val)
        }
        else {
          breaks <- Insert((breaks[k]+ breaks[k+1])/2, breaks, k)
          counts <- Insert(val, counts, k)
        }
      }
   NrB = NrB + 1
 }
```

Fig. 1. The PiD algorithm for updating $layer_1$.

equal-width strategy. Each time we observe a value of the random variable, we update $layer_1$. The update process determines the interval corresponding to the observed value, and increments the count of this interval. Whenever the count of an interval is above a user defined threshold (a percentage of the total number of points seen so far), a split operator triggers. The split operator generates new intervals in $layer_1$. If the interval that triggers the split operator is the first or the last a new interval with the same step is inserted. In all the other cases, the interval is split into two, generating a new interval.

The process of updating $layer_1$ works online, performing a single scan over the data stream. It can process infinite sequences of data, processing each example in constant time and space.

The second layer merges the set of intervals defined by the first layer. It triggers whenever it is necessary (e.g. by user action). The input for the second layer is the breaks and counts of $layer_1$, the type of histogram (equal-width or equal-frequency) and the number of intervals. The algorithm for the $layer_2$ is very simple. For equal-width histograms, it first computes the breaks of the final histogram, from the actual range of the variable (estimated in $layer_1$). The

algorithm traverses the vector of breaks once, adding the counts corresponding to two consecutive breaks. For equal-frequency histograms, we first compute the exact number F of points that should be in each final interval (from the total number of points and the number of desired intervals). The algorithm traverses the vector of counts of $layer_1$ adding the counts of consecutive intervals till F.

The two-layer architecture divides the histogram problem into two phases. In the first phase, the algorithm traverses the data stream and incrementally maintains an equal-with discretization. The second phase constructs the final histogram using only the discretization of the first phase. The computational costs of this phase can be ignored: it traverses once the dicretization obtained in the first phase. We can construct several histograms using different number of intervals, and different strategies: equal-width or equal-frequency. This is the main advantage of PiD in exploratory data analysis.

2.2 Analysis of the algorithm

The histograms generated by PiD are not exact. There are two sources of error:

1. The set of boundaries. The breaks of the histogram generated in the second layer are restricted to set of breaks defined in the first layer.
2. The frequency counters. The counters of the second layer are aggregations of the frequency counters in the first layer. If the splitting operator doesn't trigger, counters in first layer are exact, and also counters in second layer. The splitting operator can produce inexact counters. If the merge operation of the second layer aggregate those intervals, final counters are correct.

A comparative analysis of the histograms produced by PiD and histograms produced by exact algorithms using all the data, reveals some properties of the PiD algorithm. Assuming a equal-width discretization (that is the split operator didn't trigger) for the first layer and any method for the second layer, the error of PiD boundaries (that is the sum of absolute differences between boundaries between PiD and batch discretization) is bound, in the worst case, by: $R * N_2/(2 * N_1)$, where N_1 denotes the number of intervals of $layer_1$, N_2 the number of intervals of $layer_2$, and R is the range of the random variable. This indicates that when N_1 increases the error decreases. The algorithm guarantees that frequencies at second layer are exact (for the second layer' boundaries). We should note that the splitting operator will always decrease the error.

The time complexity of PiD depends on the discretization methods used in each layer. The time complexity of the second layer is constant because its input is the first layer that has a (almost) fixed number of intervals. The time complexity for the first layer is linear in the number of examples.

2.3 Experimental Evaluation

In this section we compare the output of PiD against the set of intervals obtained using all the training data and the same discretization strategy of PiD. For

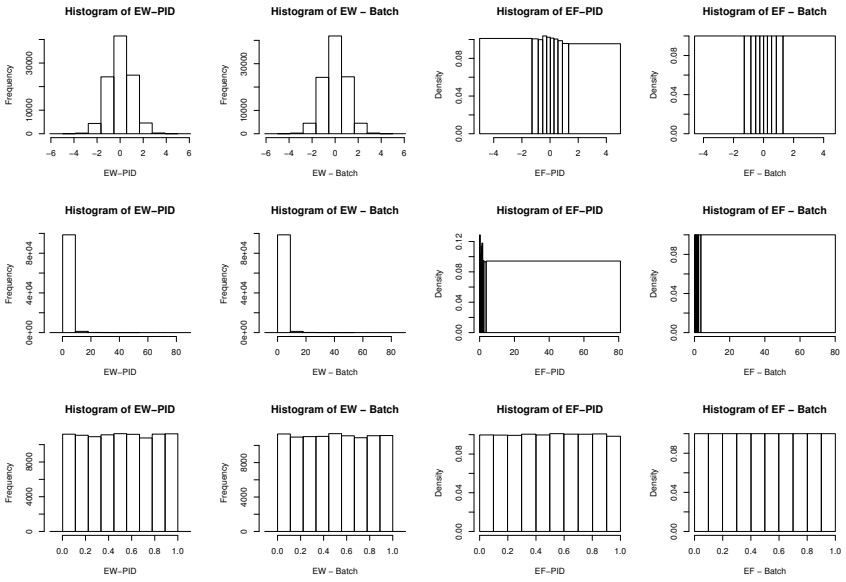

Fig. 2. Comparison between histograms using PiD and batch. We show the histograms (using equal-width and equal-frequency) for Normal, Log-Normal, and Uniform random variables.

$layer_1$, the number of intervals was set to 200. For $layer_2$, the number of intervals was set to 10, in two scenarios: equal width and equal frequency.

We evaluate two quality measures: the set of boundaries and frequencies. Boundaries are evaluated using the mean absolute deviation: $mad(P,S) = \sum(|P_i - S_i|)/n$, and the mean squared error: $mse(P,S) = \sqrt{\sum(P_i - S_i)^2}/n$. To compare frequencies, we use the affinity coefficient [3]: $AF(P,S) = \sum \sqrt{P_i \times S_i}$ [2]. Its range is $[0;1]$; values near 1 indicate that the two distributions are not different.

We have done some experiments using a sample of 100k values of a random variable from different distributions: Uniform, Normal, and Log-Normal. Results reported here is the average of 100 experiments. The median of the performance metrics are presented in table 1. The discretization provided by $layer_2$ is based on the summaries provided by $layer_1$. Of course, the summarization of $layer_1$ introduces an error. These results provide evidence that the impact of this error in the final discretization is reduced. Moreover, in a set of similar experiments, we have observed that increasing the number of intervals of $layer_1$ decreases the error.

The advantage of the two-layer architecture of PiD, is that after generating the $layer_1$, the computational costs, in terms of memory and time, to generate

[2] P denotes the set of boundaries (frequency in the case of AF) defined by PiD, and S denotes the set of boundaries (frequency) defined using all the data.

Equal-Frequency	Affinity	MAD	MSE	**Equal-Width**	Affinity	MAD	MSE
Normal	0.999	0.128	0.084	Uniform	1.000	0.010	0.004
Lognormal	0.995	0.086	0.047	Lognormal	1.000	0.081	0.029
Uniform	1.000	0.001	0.004	Normal	0.999	0.002	0.001

Table 1. Average results of evaluation metrics of the quality of discretization.

the final histogram (the $layer_2$) is low: only depends on the number of intervals of the $layer_1$. From $layer_1$ we can generate histograms with different number of intervals and using different strategies (equal-width or equal-frequency). We should note that the standard algorithm to generate equal-frequency histograms requires a sort operation, that could be costly for large n. This is not the case of PiD. Generation of equal-frequency histograms from the $layer_1$ is straightforward.

3 Applications to Data Mining

Discretization of continuous attributes is an important task for certain types of machine learning algorithms. Bayesian approaches, for instance, require assumptions about data distribution. Decision Trees require sorting operations to deal with continuous attributes, which largely increase learning times. In the next subsection we revisit the most common methods used for discretization. Nowadays there are hundreds of discretization methods. In [7, 16, 8, 14] the authors present a large overview of the area.

3.1 Review of discretization methods

Dougherty *et al.* [7] defined three different axis upon which we may classify discretization methods: *supervised vs. unsupervised*, *global vs. local* and *static vs. dynamic*. *Supervised methods* use the information of class labels while *unsupervised methods* do not. *Local methods* like the one used by C4.5, produce partitions that are applied to localized regions of the instance space. *Global methods* such as binning are applied before the learning process to the whole data. In *static methods* attributes are discretized independently of each other, while *dynamic methods* take into account the interdependencies between them. Note that in this paper we will review just *static methods*. There is another axis, *parametric* vs. *non-parametric* methods, where *parametric* methods need user input like the number of cut-points and *non-parametric* uses only information from data.

Some examples of discretization methods are:

- **Equal width discretization (EWD)**. It is the simplest method to discretize a continuous-valued attribute, if divides the range of observed values for a feature into k equally sized bins. The main advantaged of this *global*, *unsupervised*, and *parametric* method is its simplicity, but has a major drawback, the presence of outliers.

- **Equal frequency discretization (EFD)**. Like the previous method, it divides the range of observed values into k bins where (considering n instances) each bin contains n/k values. This method is also *global*, *unsupervised* and *parametric* for the same reasons.
- **k-means**. It is a more sophisticated version of bins. In this method the distribution of the values over the k intervals minimizes the intra-interval and maximizes the inter-interval distance. This is an iterative method that begins with an equal with discretization, iteratively adjust the boundaries to minimize a squared-error function and only stops when it can not change any value to improve the criteria above. Like the other methods referred here, this is a *global*, *unsupervised* and *parametric* method.
- **Recursive entropy discretization (ENT-MDL)** presented by Fayyad and Irani [9] in 1993. This algorithm uses the class information entropy of candidate partitions to select the boundaries for discretization. It starts finding a single threshold that minimizes the entropy function over all possible cut-points; it is then recursively applied to both of the partitions. The stopping criteria uses the *minimum description length* principle [15]. We refer to this algorithm as Ent-MDL [7]. It is used in the experimental evaluation section. This is a *supervised*, *non-parametric*, and *global* method.
- **Proportional discretization (PD)**, was presented by Ying Yang [16]. It calculates the number of intervals and frequency with a very simple formula, $s \times t = n$, where t is the number of intervals, s is the frequency in each interval and n the number of training instances. This method was created in the context of naive Bayes. The author argues that by setting the number of intervals and frequency proportional to the training data reduces the *bias* and *variance*. This is also a more sophisticated version of bins. It uses no class information and it is for all data set, so it is a *global*, *unsupervised* and *non-parametric* method.

3.2 Applying PiD in Supervised Learning

The application of the framework defined in section 2.1 to classification supervised learning is straightforward. We only need to consider another data structure, in both layers, to store the information about the number of examples per class in each interval. For both layers, the data structure is a matrix where the number of columns is equal to the number of intervals and the number of rows is equal to the number of classes. To update the distribution matrix in $layer_1$ we need the class value associated with each example.

From the information stored in the distribution matrix it is easy to obtain the conditional probability that an attribute-value belongs to an interval given that the corresponding example belongs to a class: $P(b_i < x \leq b_{i+1}|Class_j)$. This is the information required by some classification algorithms, for example decision trees using multi-interval discretization and naive Bayes. It is relevant to note that PiD gives for free all the information required by naive Bayes. This is why we use this algorithm in our experimental section.

So far we have seen two unsupervised discretization methods: equal width and equal frequency. The distribution matrix in $layer_1$ allow us to use supervised discretization methods like *Recursive entropy discretization* [10] or *chi-merge* [13, 4]. These methods have some advantages: they are supervised, so more appropriate for classification problems, and the number of intervals of the final discretization ($layer_2$ in our case) is automatically determined. We have implemented and used in the experimental section the Recursive entropy discretization.

In some learning scenarios, for example online learning, we need to maintain a permanent discretization in $layer_2$. Till now, $layer_1$ is updated online but $layer_2$ is computed from time to time when requested. Suppose we have a $layer_1$ and a $layer_2$. Whenever a new example is available we update $layer_1$ (and the distribution matrix) using the algorithm in figure 1. It is improbable that a single observation changes the distribution in $layer_2$, so we find the corresponding interval in $layer_2$ and update the frequency counts and distribution matrix. Now we are faced with the problem: When should we reconstruct $layer_2$? We consider three cases:

- Equal-width histograms are defined by breaks b_i such that $b_i - b_{i+1}$ is constant for each i. The $layer_2$ needs to be reconstructed only when the set of breaks in $layer_1$ changes. This only occurs whenever the split operator triggers. After applying the split operator the algorithm reconstruct $layer_2$.
- Equal-frequency histograms are defined by frequencies f_i. PiD does not guarantee that all f_i are equal. Suppose that after seeing n examples, the algorithm reconstruct $layer_2$. We define two thresholds $T_{min} = min(f_i/n)$ and $T_{max} = max(f_i/n)$. Whenever we observe an interval with frequency below $(1 - \alpha) \times T_{min}$ or above $(1 + \alpha) \times T_{max}$ we reconstruct $layer_2$ [3].
- Other type of histograms, for example, entropy based or chi-merge, we haven't find well founded strategies. We use a very simple strategy that reconstructs $layer_2$ after seeing a pre-defined number of examples.

Most of naive Bayes implementations assume a pre-defined number of attribute-values for each attribute. The recursive entropy discretization can generate for each attribute different numbers of intervals at different times. The information provided by PiD allow us to use naive-Bayes in these cases. Another issue is: knowing the range of the random variable and fixing the number of intervals, equal-width discretization is naturally incremental. Why should we use PiD with equal-width in $layer_2$? What is the advantage? The advantage is that we can use different number of intervals in $layer_2$. Most of the rules for determining the number of intervals point out that for increasing number of examples we should use increasing number of intervals. Its straightforward to incorporate this ability in PiD.

[3] A reasonable value for α seems to be 1%.

	Examples	Attributes	Classes	Frequency	Entropy	Width	NB-Weka
Iris	150	4	3	5.99±0.1	4.00±0.1	4.0±0.1	3.92±.02
Optdigits	5053	64	10	20.41±.04	8.85±.02	21.23±.02	8.21±0.1
Satimage	6435	36	6	19.53±.07	**18.93±.07**	20.02±.02	20.32±0.1
Segmentation	2310	19	7	10.74±.02	**7.61±.02**	10.82±.02	19.44±0.0
Shuttle	58000	9	7	0.417±.04	**0.474±.00**	8.57±.00	6.98±0.1
Wine	178	13	3	4.54±.04	1.66±.04	3.33±.04	1.73±0.1
Average Error				10.27	7.25	11.33	10.05
Average Rank				2.66	1.83	3.16	2.33

Table 2. Results of PiD variants and naive Bayes (Weka, assuming a normal distribution) in benchmark classification problems.

4 Experimental Evaluation

4.1 Methodology

We have evaluated the naive Bayes algorithm described in section 3.2 in a set of benchmark data sets are from the U. C. Irvine repository [2]. For comparative purposes we present the results of a standard naive Bayes from WEKA [15][4]. Accuracy was evaluated using 10-fold cross validation. All algorithms generate a model on the same training sets and models were evaluated on the same test sets. For PiD, we carried out experiments with supervised and unsupervised discretization in the second layer. The supervised discretization method chosen was the *recursive entropy discretization*. This method is shortly described in this paper, but more information about the method can be found in [9]. We use also two unsupervised discretization methods *equal-frequency* and *equal-width*. PiD was used with 200 intervals in $layer_1$ and with 40 intervals in $layer_2$.

Table 2 presents the results. Entropy based discretization performs quite well mostly in large datasets. PiD unsupervised methods are competitive against a standard naive-Bayes algorithm. Significant differences from Wilcoxon test (confidence level 95%) between Entropy and NB-Weka are shown in bold. These results illustrate that PiD is competitive against standard methods in benchmark problems. Nevertheless, the main advantages of PiD are low-memory usage and processing time, that can not be clear illustrate in these datasets. The nest subsection presents an example that clear illustrates these advantages.

4.2 A real-world Application

In the dataset *trawl hauls* (Fig. 3) the goal is to predict when a trawl vessel is fishing or not from GPS information [1]. The attributes are derived from a window of 11 consecutive points (five before the decision point and 5 after). This is a two class problem, defined by 22 attributes. 11 attribures are the direction and the other 11 are the speed of the vessel. From all the available data, 675.329

[4] The version used assumes a normal distribution for the continuous attributes. It is used because it is also incremental.

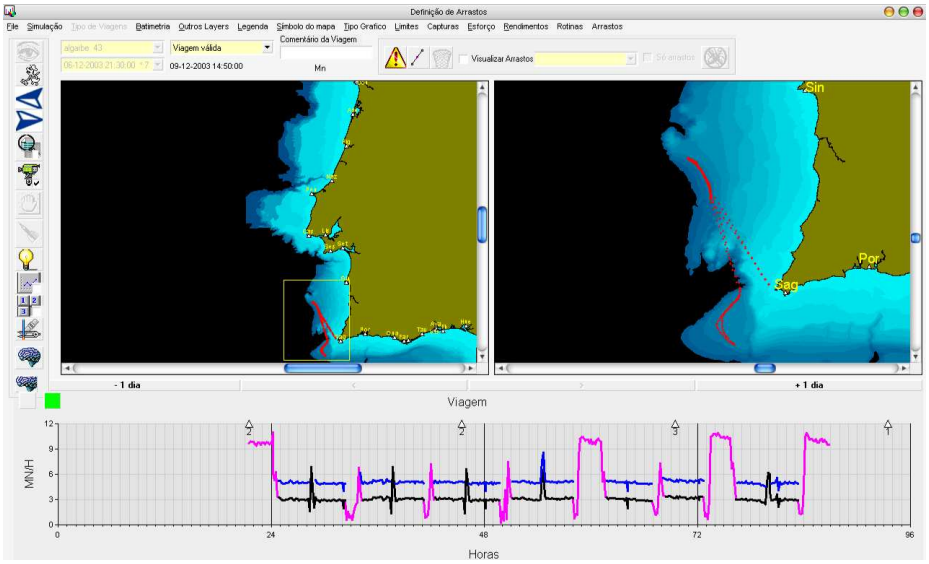

Fig. 3. Illustrative figure for *trawl haul* problem. The bottom panel shows the speed of the vessel and predicted trawl hauls. The blue lines show the predictions and the black lines show the trawl hauls identified by the user.

examples, we randomly select 10% for test and the remainder (607886 examples) were used for training. Using this training set and test set the results of the 3 PiD discretization methods using a naive Bayes classifier are: Entropy: 6.86%, Equal-Width: 6.93%, and Equal-frequency: 6.91%. For comparative purposes the error rate of naive Bayes updatable in WEKA [15] is 9.11% [5]. The Figure 4 points out relevant information. The left figure plots the error rate of varying the number of intervals in $layer_2$ using equal-width and equal-frequency discretization[6]. For both methods, the best results were obtained using 40 intervals. Its interesting to observe that using few intervals has strong impact in the performance of equal-width discretization, while too much intervals has strong impact in the performance of equal-frequency discretization. The right figure plots learning curves using the 3 discretization methods (for the parametric methods the number of intervals of $layer_2$ was set to 40). In all cases, the error decreases when increasing the number of examples. The minimum error corresponds to 450K examples. Up to 200K examples the best results were obtained using equal-frequency discretization. For more than 200K examples entropy discretization dominates the other two methods.

[5] Other variants, including the discretization procedures implemented in Weka didn't run due to lack of memory.

[6] The initial number of intervals for $layer_1$ was fixed to 500. Varying the number of intervals in this layer has almost no impact in the final results.

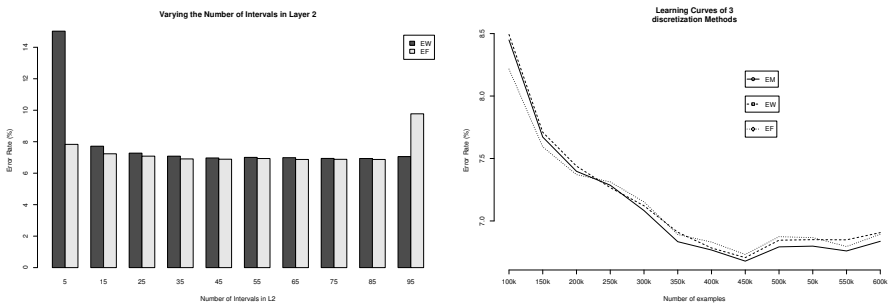

Fig. 4. Sensitivity analysis in *Arrastos* data. (a) Varying the number of intervals in $layer_2$. (b) Learning curves using the 3 discretization methods (EW,EF, and Entropy).

5 Conclusion and Future Work

In this paper we present a new method for incremental discretization. Although incremental learning is a hot topic in machine learning and discretization is a fundamental pre-processing step for some well-known algorithms, the topic of incremental discretization has received few attention from the community. The work on data synopsis and histograms of Guha [12] and Yossi [11] and the work on symbolic representation for time series (Keogh [5]) are somewhat related to the work presented here. An advantage of PiD is the ability of using flexible number of intervals in the final discretization.

We have introduced a new discretization method that works in two layers. This two-stage architecture is very flexible. It can be used as supervised or unsupervised modes. For the second layer any base discretization method can be used: *equal frequency*, *recursive entropy discretization*, *chi-merge*, etc. The most relevant aspect is that the boundaries and the number of intervals in both layers can change when new data is available. PiD processes training examples in a single scan over the data. It process examples in constant time and space even for infinite sequences of streaming data. The main advantage of the proposed method is the use of (almost) constant memory even in the presence of infinite sequences of data. Since discretization simplifies and reduce data space, having a first layer with *equal width* as preparation for a second layer with methods more complex like *recursive entropy discretization* or *chi-merge* can be faster than applying the methods to the whole data set.

The method described here can be used as pre-processing to other learning algorithms that requires discretized data, for example, Bayesian Networks, Association Rules, etc. Another interesting research goal is discretization in dynamic environments with concept drift. We believe that online algorithms that monitor changes in distributions in the target variable can provide useful results.

Acknowledgments

Thanks to the financial support given by the FEDER, the Plurianual support attributed to LIACC, project ALES II (POSI/EIA/55340/2004), and Mario Alves for revising a previous version of the paper.

References

1. M. Afonso-Dias, J. Simoes, and C. Pinto. Gis/spatial analysis in fishery and aquatic sciences. In *Proceedings 2th International Symposium on GIS/Spatial Analysis in Fishery and Aquatic Sciences*, pages 323–340. Saitama, Japan, 2004.
2. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
3. H. Bock and E. Diday. *Analysis of symbolic data: Exploratory Methods for Extracting Statistical Information from Complex Data*. Springer Verlag, 2000.
4. Marc Boulle. Khiops: A statistical discretization method of continuous attributes. *Machine Learning*, 55(1):53–69, 2004.
5. B. Chiu, E. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *Proceedings 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 493–498. ACM Press, 2003.
6. Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.
7. James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings 12th International Conference on Machine Learning*, pages 194–202. Morgan and Kaufmann, 1995.
8. Tapio Elomaa and Juho Rousu. Necessary and suficient pre-processing in numerical range discretization. *Knowledge and Information Systems (2003) 5: 162 182*, 5:162–182, 2003.
9. U. M. Fayyad and K. B Irani. Multi-interval discretization of continuous valued attributes for classification learning. In *13 International Joint Conference on Artificial Intelligence*, pages 1022–1027. Morgan Kaufmann, 1993.
10. Usama M. Fayyad and Keki B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87–102, 1992.
11. P. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *ACM Transactions on Database Systems*, 5:1–33, 2002.
12. Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and histograms. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 471–475. ACM Press, 2001.
13. R. Kerber. Chimerge discretization of numeric attributes. In *Proceeding of the 10th International Conference on Artificial Intelligence*, pages 123–128, 1991.
14. M. J. Pazzani. An iterative improvement approach for the discretization of numeric attributes in bayesian classifiers. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, 1995.
15. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers, 1999.
16. Ying Yang. *Discretization for Naive-Bayes Learning*. PhD thesis, School of Computer Science and Software Engineering of Monash University, July 2003.